Progress®



ABL Reference

OpenEdge®

Copyright

© **2021 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.**

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Defrag This, Deliver More Than Expected, DevReach (and design), Icenium, Inspec, Ipswitch, iMacros, Kendo UI, Kinvey, MessageWay, MOVEit, NativeChat, NativeScript, OpenEdge, Powered by Chef, Powered by Progress, Progress, Progress Software Developers Network, SequeLink, Sitefinity (and Design), Sitefinity, Sitefinity (and design), SpeedScript, Stylus Studio, Stylized Design (Arrow/3D Box logo), Styleized Design (C Chef logo), Stylized Design of Samurai, TeamPulse, Telerik, Telerik (and design), Test Studio, WebSpeed, WhatsConfigured, WhatsConnected, WhatsUp, and WS_FTP are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries.

Analytics360, AppServer, BusinessEdge, Chef Automate, Chef Compliance, Chef Desktop, Chef Habitat, Chef WorkStation, Corticon.js, Corticon Rules, Data Access, DataDirect Autonomous REST Connector, DataDirect Spy, DevCraft, Fiddler, Fiddler Everywhere, FiddlerCap, FiddlerCore, FiddlerScript, Hybrid Data Pipeline, iMail, JustAssembly, JustDecompile, JustMock, KendoReact, NativeScript Sidekick, OpenAccess, PASOE, Pro2, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, Push Jobs, SafeSpaceVR, Sitefinity Cloud, Sitefinity CMS, Sitefinity Digital Experience Cloud, Sitefinity Feather, Sitefinity Insight, Sitefinity Thunder, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Supermarket, SupportLink, Unite UX, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

July 2021

Product version: Progress OpenEdge 12.4

Updated: 2021/06/27

Table of Contents

Introduction to the ABL Reference.....59

ABL Syntax Reference.....61

: Punctuation.....61
:: Punctuation.....62
. Punctuation.....62
; Punctuation.....63
, Punctuation.....63
; Special character.....63
? Special character.....63
\ Special character.....64
~ Special character.....65
" Special character.....66
' Special character.....66
/ Special character.....66
( ) Expression precedence.....66
[ ] Array reference.....67
= Special character.....67
< Special character.....67
< = Special character.....67
< > Special character.....67
> Special character.....67
> = Special character.....67
" "Character-string literal.....67
{ } Argument reference.....68
{ } Include file reference.....70
{ } Preprocessor name reference.....74
&GLOBAL-DEFINE preprocessor directive.....78
&IF, &THEN, &ELSEIF, &ELSE, and &ENDIF preprocessor directives.....79
&MESSAGE preprocessor directive.....82
&SCOPED-DEFINE preprocessor directive.....83
&UNDEFINE preprocessor directive.....84
/* Multi-line comments */.....84
// Single-line comments.....85
+ Unary positive operator.....86
+ Addition operator.....87
+ Concatenation operator.....88
+ Date addition operator.....89

+ Datetime addition operator....90
+= Addition assignment operator....91
- Unary negative operator....94
- Subtraction operator....94
- Date subtraction operator....95
- Datetime subtraction operator....96
-= Subtraction assignment operator....98
* Multiplication operator....100
*= Multiplication assignment operator....101
/ Division operator....102
/= Division assignment operator....103
ABSOLUTE function....105
ACCUM function....106
ACCUMULATE statement....107
ACTIVE-FORM system reference....109
ADD-INTERVAL function....110
Aggregate phrase....111
AGGREGATE statement....113
ALIAS function....115
AMBIGUOUS function....116
AND operator (bitwise)....117
AND operator (logical)....118
APPLY statement....119
ASC function....121
Assignment (=) statement....122
ASSIGN statement....126
AT phrase....131
AUDIT-ENABLED function....134
AVAILABLE function....135
BASE64-DECODE function....136
BASE64-ENCODE function....137
BEGINS operator....138
BELL statement....139
BLOCK-LEVEL ON ERROR UNDO, THROW statement....140
BOX function....142
BUFFER-COMPARE statement....144
BUFFER-COPY statement....146
BUFFER-GROUP-ID function....148
BUFFER-GROUP-NAME function....149
BUFFER-PARTITION-ID function....149
BUFFER-TENANT-ID function....150
BUFFER-TENANT-NAME function....151
CALL statement....151
CAN-DO function....152
CAN-FIND function....155

CAN-QUERY function....158
CAN-SET function....159
CAPS function....160
CASE statement....161
CAST function....163
CATCH statement....164
CHOOSE statement....168
CHR function....173
Class-based data member access....174
Class-based method call....177
Class-based object reference....180
Class-based property access....181
CLASS statement....185
CLEAR statement....204
CLOSE QUERY statement....205
CLOSE STORED-PROCEDURE statement....207
CODEPAGE-CONVERT function....208
COLOR phrase....210
COLOR statement....213
COMBO-BOX phrase....215
COMPARE function....220
COMPILE statement....222
CONNECT statement....253
CONNECTED function....259
CONSTRUCTOR statement....260
COPY-LOB statement....264
COUNT-OF function....268
CREATE statement....268
CREATE ALIAS statement....271
CREATE automation object statement....274
CREATE BROWSE statement....278
CREATE BUFFER statement....281
CREATE CALL statement....283
CREATE CLIENT-PRINCIPAL statement....284
CREATE DATABASE statement....285
CREATE DATASET statement....287
CREATE DATA-SOURCE statement....288
CREATE QUERY statement....289
CREATE SAX-ATTRIBUTES statement....290
CREATE SAX-READER statement....293
CREATE SAX-WRITER statement....294
CREATE SERVER statement....294
CREATE SERVER-SOCKET statement....295
CREATE SOAP-HEADER statement....295
CREATE SOAP-HEADER-ENTRYREF statement....296

CREATE SOCKET statement....297
CREATE TEMP-TABLE statement....297
CREATE widget statement....299
CREATE WIDGET-POOL statement....302
CREATE X-DOCUMENT statement....304
CREATE X-NODEREF statement....305
CURRENT-CHANGED function....306
CURRENT-LANGUAGE function....308
CURRENT-LANGUAGE statement....309
CURRENT-RESULT-ROW function....310
CURRENT-VALUE function....311
CURRENT-VALUE statement....313
Data types....315
DATASERVERS function....334
DATA-SOURCE-MODIFIED function....335
DATE function....335
DATETIME function....338
DATETIME-TZ function....340
DAY function....342
DBCODEPAGE function....343
DBCOLLATION function....344
DBNAME function....345
DBPARAM function....346
DB-REMOTE-HOST function....347
DBRESTRICTIONS function....347
DBTASKID function....349
DBTYPE function....350
DBVERSION function....351
DDE ADVISE statement....352
DDE EXECUTE statement....354
DDE GET statement....356
DDE INITIATE statement....358
DDE REQUEST statement....359
DDE SEND statement....361
DDE TERMINATE statement....363
DECIMAL function....364
DECRYPT function....365
DEFINE BROWSE statement....366
DEFINE BUFFER statement....380
DEFINE BUTTON statement....386
DEFINE DATASET statement....393
DEFINE DATA-SOURCE statement....401
DEFINE ENUM statement....404
DEFINE EVENT statement....404
DEFINE FRAME statement....410

DEFINE IMAGE statement....419
DEFINE MENU statement....423
DEFINE PARAMETER statement....429
DEFINE PROPERTY statement....442
DEFINE QUERY statement....460
DEFINE RECTANGLE statement....465
DEFINE STREAM statement....469
DEFINE SUB-MENU statement....471
DEFINE TEMP-TABLE statement....477
DEFINE VARIABLE statement....490
DEFINE WORK-TABLE statement....504
DEFINE WORKFILE statement....509
DEFINED preprocessor function....509
DELETE statement....510
DELETE ALIAS statement....513
DELETE OBJECT statement....513
DELETE PROCEDURE statement....517
DELETE WIDGET statement....519
DELETE WIDGET-POOL statement....521
DESTRUCTOR statement....523
DICTIONARY statement....524
DISABLE statement....525
DISABLE TRIGGERS statement....528
DISCONNECT statement....530
DISPLAY statement....531
DO statement....537
DOS statement....543
DOWN statement....544
DYNAMIC-CAST function....546
DYNAMIC-CURRENT-VALUE function....548
DYNAMIC-CURRENT-VALUE statement....549
DYNAMIC-ENUM function....550
DYNAMIC-FUNCTION function....551
DYNAMIC-INVOKE function....553
DYNAMIC-NEW statement....555
DYNAMIC-NEXT-VALUE function....558
DYNAMIC-PROPERTY function....559
DYNAMIC-PROPERTY statement....560
EDITING phrase....562
EDITOR phrase....564
EMPTY TEMP-TABLE statement....567
ENABLE statement....568
ENCODE function....573
ENCRYPT function....574
END statement....575

ENTERED function....576
ENTRY function....577
ENTRY statement....579
ENUM statement....581
Enumeration member access....589
EQ or = operator....590
ERROR function....591
ETIME function....592
EXP function....593
EXPORT statement....594
Expression....597
EXTENT function....602
EXTENT statement....603
FILL function....605
FINALLY statement....606
FIND statement....609
FIRST function....618
FIRST-OF function....619
FIX-CODEPAGE statement....619
FOR statement....620
FORM statement....632
Format phrase....638
Frame phrase....647
FRAME-COL function....660
FRAME-DB function....661
FRAME-DOWN function....662
FRAME-FIELD function....663
FRAME-FILE function....664
FRAME-INDEX function....664
FRAME-LINE function....665
FRAME-NAME function....667
FRAME-ROW function....668
FRAME-VALUE function....669
FRAME-VALUE statement....670
FUNCTION statement....671
GATEWAYS function....678
GE or >= operator....678
GENERATE-PBE-KEY function....679
GENERATE-PBE-SALT function....680
GENERATE-RANDOM-KEY function....681
GENERATE-UUID function....681
GET statement....682
GET-BITS function....684
GET-BYTE function....685
GET-BYTE-ORDER function....686

GET-BYTES function....686
GET-CLASS function....687
GET-CODEPAGE function....688
GET-CODEPAGES function....688
GET-COLLATION function....689
GET-COLLATIONS function....689
GET-DB-CLIENT function....690
GET-DOUBLE function....692
GET-EFFECTIVE-TENANT-ID function....693
GET-EFFECTIVE-TENANT-NAME function....693
GET-FLOAT function....694
GET-INT64 function....695
GET-KEY-VALUE statement....696
GET-LONG function....699
GET-POINTER-VALUE function....699
GET-SHORT function....701
GET-SIZE function....701
GET-STRING function....703
GET-UNSIGNED-LONG function....703
GET-UNSIGNED-SHORT function....704
GO-PENDING function....705
GT or > operator....706
GUID function....707
HANDLE function....708
HEX-DECODE function....709
HEX-ENCODE function....709
HIDE statement....710
IF...THEN...ELSE function....712
IF...THEN...ELSE statement....713
Image phrase....715
IMPORT statement....717
INDEX function....722
INPUT function....724
INPUT CLEAR statement....725
INPUT CLOSE statement....726
INPUT FROM statement....727
INPUT THROUGH statement....733
INPUT-OUTPUT CLOSE statement....737
INPUT-OUTPUT THROUGH statement....738
INSERT statement....742
INT64 function....744
INTEGER function....746
INTERFACE statement....747
INTERVAL function....757
IS-ATTR-SPACE function....758

IS-CODEPAGE-FIXED function....758
IS-COLUMN-CODEPAGE function....758
IS-DB-MULTI-TENANT function....759
IS-LEAD-BYTE function....759
ISO-DATE function....760
KBLABEL function....760
KEYCODE function....761
KEYFUNCTION function....762
KEYLABEL function....764
KEYWORD function....764
KEYWORD-ALL function....766
LAST function....767
LASTKEY function....767
LAST-OF function....769
LC function....769
LDBNAME function....770
LE or < = operator....772
LEAVE statement....773
LEFT-TRIM function....774
LENGTH function....775
LENGTH statement....777
LIBRARY function....778
LINE-COUNTER function....779
LIST-EVENTS function....781
LIST-QUERY-ATTRS function....782
LIST-SET-ATTRS function....783
LIST-WIDGETS function....784
LOAD statement....785
LOAD-PICTURE statement....788
LOCKED function....789
LOG function....790
LOGICAL function....791
Logical values....792
LOOKUP function....793
LT or < operator....794
MATCHES operator....795
MAXIMUM function....796
MD5-DIGEST function....797
MEMBER function....798
MESSAGE statement....799
MESSAGE-DIGEST function....806
MESSAGE-LINES function....807
METHOD statement....808
MINIMUM function....816
MODULO operator....817

MONTH function....818
MTIME function....819
NE or <> operator....820
NEW function (classes)....821
NEW function (record buffers)....825
NEW statement....826
NEXT statement....828
NEXT-PROMPT statement....828
NEXT-VALUE function....830
NORMALIZE function....832
NOT operator (bitwise)....832
NOT operator (logical)....833
NOT ENTERED function....834
NOW function....835
NO-ERROR option....835
NUM-ALIASES function....836
NUM-DBS function....837
NUM-ENTRIES function....837
NUM-RESULTS function....839
ON ENDKEY phrase....841
ON ERROR phrase....843
ON QUIT phrase....846
ON statement....848
ON STOP phrase....854
OPEN QUERY statement....857
OPSYS function....862
OR operator (bitwise)....863
OR operator (logical)....863
OS-APPEND statement....864
OS-COMMAND statement....866
OS-COPY statement....867
OS-CREATE-DIR statement....868
OS-DELETE statement....869
OS-DRIVES function....871
OS-ERROR function....871
OS-GETENV function....873
OS-RENAME statement....874
OUTPUT CLOSE statement....875
OUTPUT THROUGH statement....876
OUTPUT TO statement....880
OVERLAY statement....888
PAGE statement....890
PAGE-NUMBER function....891
PAGE-SIZE function....891
Parameter definition syntax....892

Parameter passing syntax....898
PAUSE statement....912
PDBNAME function....914
PRESELECT phrase....914
PROC-HANDLE function....917
PROC-STATUS function....918
PROCEDURE statement....919
PROCESS-ARCHITECTURE function....924
PROCESS EVENTS statement....925
PROGRAM-NAME function....926
PROGRESS function....928
PROMPT-FOR statement....929
PROMSGS function....936
PROMSGS statement....936
PROPATH function....937
PROPATH statement....938
PROVERSION function....940
PUBLISH statement....941
PUT CURSOR statement....944
PUT SCREEN statement....946
PUT statement....948
PUT-BITS statement....952
PUT-BYTE statement....953
PUT-BYTES statement....955
PUT-DOUBLE statement....955
PUT-FLOAT statement....956
PUT-INT64 statement....957
PUT-KEY-VALUE statement....958
PUT-LONG statement....962
PUT-SHORT statement....963
PUT-STRING statement....964
PUT-UNSIGNED-LONG statement....965
PUT-UNSIGNED-SHORT statement....966
QUERY-OFF-END function....966
QUERY-TUNING phrase....967
QUIT statement....970
QUOTER function....971
R-INDEX function....973
RADIO-SET phrase....975
RANDOM function....977
RAW function....978
RAW statement....979
RAW-TRANSFER statement....980
READKEY statement....982
RECID function....984

Record phrase....985
RECORD-LENGTH function....996
REJECTED function....997
RELEASE statement....997
RELEASE EXTERNAL statement....999
RELEASE OBJECT statement....999
REPEAT statement....1001
REPLACE function....1007
REPOSITION statement....1008
RETRY function....1011
RETURN statement....1012
RETURN-VALUE function....1015
RGB-VALUE function....1015
RIGHT-TRIM function....1016
ROUND function....1018
ROUTINE-LEVEL ON ERROR UNDO, THROW statement....1018
ROW-STATE function....1020
ROWID function....1021
RUN statement....1023
RUN STORED-PROCEDURE statement....1041
RUN SUPER statement....1044
SAVE CACHE statement....1047
SCREEN-LINES function....1049
SCROLL statement....1049
SDBNAME function....1055
SEARCH function....1056
SEEK function....1058
SEEK statement....1059
SELECTION-LIST phrase....1061
SET statement....1064
SET-BYTE-ORDER statement....1071
SET-DB-CLIENT function....1072
SET-EFFECTIVE-TENANT function....1076
SET-POINTER-VALUE statement....1077
SET-SIZE statement....1078
SETUSERID function....1080
SHA1-DIGEST function....1082
SHOW-STATS statement....1083
SIZE phrase....1084
SLIDER phrase....1085
SQRT function....1088
SSL-SERVER-NAME function....1089
STATUS statement....1089
STOP statement....1091
STOP-AFTER phrase....1092

STRING function....1094
SUBSCRIBE statement....1096
SUBSTITUTE function....1098
SUBSTRING function....1099
SUBSTRING statement....1100
SUPER function....1103
SUPER statement....1104
SUPER system reference....1105
SYSTEM-DIALOG COLOR statement....1106
SYSTEM-DIALOG FONT statement....1107
SYSTEM-DIALOG GET-DIR statement....1109
SYSTEM-DIALOG GET-FILE statement....1110
SYSTEM-DIALOG PRINTER-SETUP statement....1114
SYSTEM-HELP statement....1115
TENANT-ID function....1120
TENANT-NAME function....1121
TENANT-NAME-TO-ID function....1121
TERMINAL function....1122
TERMINAL statement....1123
THIS-OBJECT statement....1124
THIS-OBJECT system reference....1124
TIME function....1126
TIMEZONE function....1127
TODAY function....1127
TO-ROWID function....1128
TRANSACTION function....1130
TRANSACTION-MODE AUTOMATIC statement....1130
Trigger phrase....1131
TRIGGER PROCEDURE statement....1133
TRIM function....1137
TRUNCATE function....1139
Type-name syntax....1140
TYPE-OF function....1146
UNBOX function....1146
UNDERLINE statement....1148
UNDO statement....1149
UNIX statement....1152
UNLOAD statement....1154
UNSUBSCRIBE statement....1155
UP statement....1156
UPDATE statement....1157
USE statement....1164
USERID function....1165
USING statement....1167
VALID-EVENT function....1171

VALID-HANDLE function....1171
VALID-OBJECT function....1173
VALIDATE statement....1173
VAR statement....1174
VIEW statement....1178
VIEW-AS phrase....1180
WAIT-FOR statement (ABL only)....1186
WAIT-FOR statement (.NET and ABL)....1190
WEEKDAY function....1199
WIDGET-HANDLE function....1200
Widget phrase....1201
Working directory....1203
XOR operator (bitwise)....1204
YEAR function....1205

Widget Reference....1207

BROWSE widget....1208
BUTTON widget....1218
COMBO-BOX widget....1220
CONTROL-FRAME widget....1222
DIALOG-BOX widget....1225
EDITOR widget....1227
FIELD-GROUP widget....1230
FILL-IN widget....1231
FRAME widget....1233
IMAGE widget....1237
LITERAL widget....1238
MENU widget....1239
MENU-ITEM widget....1240
RADIO-SET widget....1241
RECTANGLE widget....1243
SELECTION-LIST widget....1244
SHADOW-WINDOW widget....1246
SLIDER widget....1247
SUB-MENU widget....1249
TEXT widget....1250
TOGGLE-BOX widget....1251
WINDOW widget....1253

Handle Reference....1257

ACTIVE-WINDOW system handle....1258
Asynchronous request object handle....1259
AUDIT-CONTROL system handle....1261

AUDIT-POLICY system handle....1262
Buffer object handle....1263
Buffer-field object handle....1265
Call object handle....1266
Client-principal object handle....1273
CLIPBOARD system handle....1279
CODEBASE-LOCATOR system handle....1284
COLOR-TABLE system handle....1285
COM-SELF system handle....1288
COMPILER system handle....1288
CURRENT-WINDOW system handle....1290
Data-relation object handle....1291
Data-source object handle....1292
DEBUGGER system handle....1293
DEFAULT-WINDOW system handle....1296
DSLOG-MANAGER system handle....1297
ERROR-STATUS system handle....1299
FILE-INFO system handle....1302
FOCUS system handle....1304
FONT-TABLE system handle....1305
LAST-EVENT system handle....1307
LOG-MANAGER system handle....1309
Procedure object handle....1309
ProDataSet object handle....1311
PROFILER system handle....1313
Query object handle....1315
RCODE-INFO handle....1316
SAX-attributes object handle....1318
SAX-reader object handle....1319
SAX-writer object handle....1320
SECURITY-POLICY system handle....1321
SELF system handle....1323
Server object handle....1325
Server socket object handle....1326
SESSION system handle....1327
SOAP-fault object handle....1332
SOAP-fault-detail object handle....1333
SOAP-header object handle....1334
SOAP-header-entryref object handle....1335
Socket object handle....1336
SOURCE-PROCEDURE system handle....1337
Stream object handle....1340
TARGET-PROCEDURE system handle....1340
Temp-table object handle....1343
THIS-PROCEDURE system handle....1345

Transaction object handle....1348
WEB-CONTEXT system handle....1349
X-document object handle....1350
X-noderef object handle....1351

Handle Attributes and Methods Reference....1353

Accessing handle attributes and methods....1354
Handle-based references....1354
Handle attribute access....1356
Handle method calls....1357
Chained handle references....1357
Widget color, font, and measurement values....1358
Accessing COM object properties and methods....1358
Syntax for accessing COM object properties and methods....1359
ACCELERATOR attribute....1362
ACCEPT-CHANGES( ) method....1362
ACCEPT-ROW-CHANGES( ) method....1362
ACTIVE attribute....1363
ACTOR attribute....1363
ADD-BUFFER( ) method....1364
ADD-CALC-COLUMN( ) method....1364
ADD-COLUMNS-FROM( ) method....1366
ADD-EVENTS-PROCEDURE( ) method....1366
ADD-FIELDS-FROM( ) method....1367
ADD-FIRST( ) method....1368
ADD-HEADER-ENTRY( ) method....1369
ADD-INDEX-FIELD( ) method....1370
ADD-LAST( ) method....1370
ADD-LIKE-COLUMN( ) method....1372
ADD-LIKE-FIELD( ) method....1373
ADD-LIKE-INDEX( ) method....1374
ADD-NEW-FIELD( ) method....1374
ADD-NEW-INDEX( ) method....1376
ADD-PARENT-ID-RELATION( ) method....1377
ADD-RELATION( ) method....1377
ADD-SCHEMA-LOCATION( ) method....1379
ADD-SOURCE-BUFFER( ) method....1380
ADD-SUPER-PROCEDURE( ) method....1380
ADM-DATA attribute....1385
AFTER-BUFFER attribute....1385
AFTER-ROWID attribute....1385
AFTER-TABLE attribute....1386
ALLOW-COLUMN-SEARCHING attribute....1386
ALLOW-PREV-DESERIALIZATION attribute....1386

ALWAYS-ON-TOP attribute....1387
AMBIGUOUS attribute....1387
APPEND-CHILD( ) method....1387
APPL-ALERT-BOXES attribute....1388
APPL-CONTEXT-ID attribute....1388
APPLY-CALLBACK( ) method....1389
APPSERVER-INFO attribute....1389
APPSERVER-PASSWORD attribute....1390
APPSERVER-USERID attribute....1390
ASYNCHRONOUS attribute....1390
ASYNC-REQUEST-COUNT attribute....1391
ASYNC-REQUEST-HANDLE attribute....1391
ATTACH-DATA-SOURCE( ) method....1391
ATTACHED-PAIRLIST attribute....1392
ATTRIBUTE-NAMES attribute....1393
ATTR-SPACE attribute....1393
AUDIT-EVENT-CONTEXT attribute....1394
AUTHENTICATION-FAILED( ) method....1394
AUTO-COMPLETION attribute....1395
AUTO-DELETE attribute....1395
AUTO-DELETE-XML attribute....1396
AUTO-END-KEY attribute....1396
AUTO-GO attribute....1396
AUTO-INDENT attribute....1396
AUTO-RESIZE attribute....1397
AUTO-RETURN attribute....1398
AUTO-SYNCHRONIZE attribute....1398
AUTO-VALIDATE attribute....1398
AUTO-ZAP attribute....1399
AVAILABLE attribute....1399
AVAILABLE-FORMATS attribute....1400
BACKGROUND attribute....1400
BASE-ADE attribute....1400
BASIC-LOGGING attribute....1401
BATCH-MODE attribute....1402
BATCH-SIZE attribute....1402
BEFORE-BUFFER attribute....1402
BEFORE-ROWID attribute....1403
BEFORE-TABLE attribute....1403
BEGIN-EVENT-GROUP( ) method....1403
BGCOLOR attribute....1405
BLANK attribute....1405
BLOCK-ITERATION-DISPLAY attribute....1405
BORDER-BOTTOM-CHARS attribute....1406
BORDER-BOTTOM-PIXELS attribute....1406

BORDER-LEFT-CHARS attribute....1406
BORDER-LEFT-PIXELS attribute....1406
BORDER-RIGHT-CHARS attribute....1407
BORDER-RIGHT-PIXELS attribute....1407
BORDER-TOP-CHARS attribute....1407
BORDER-TOP-PIXELS attribute....1407
BOX attribute....1407
BOX-SELECTABLE attribute....1408
BUFFER-CHARS attribute....1408
BUFFER-COMPARE( ) method....1408
BUFFER-COPY( ) method....1410
BUFFER-CREATE( ) method....1412
BUFFER-DELETE( ) method....1412
BUFFER-FIELD attribute....1413
BUFFER-FIELD( ) method....1413
BUFFER-GROUP-ID attribute....1413
BUFFER-GROUP-NAME attribute....1414
BUFFER-HANDLE attribute....1414
BUFFER-LINES attribute....1414
BUFFER-NAME attribute....1415
BUFFER-PARTITION-ID attribute....1415
BUFFER-RELEASE( ) method....1415
BUFFER-TENANT-ID attribute....1416
BUFFER-TENANT-NAME attribute....1416
BUFFER-VALIDATE( ) method....1417
BUFFER-VALUE( ) method....1417
BYTES-READ attribute....1418
BYTES-WRITTEN attribute....1418
CACHE attribute....1419
CALL-NAME attribute....1419
CALL-TYPE attribute....1420
CANCEL-BREAK( ) method....1421
CANCEL-BUTTON attribute....1422
CANCEL-REQUESTS( ) method....1422
CANCEL-REQUESTS-AFTER( ) method....1423
CANCELLED attribute....1424
CAN-CREATE attribute....1424
CAN-DELETE attribute....1425
CAN-DO-DOMAIN-SUPPORT attribute....1425
CAN-READ attribute....1425
CAN-WRITE attribute....1426
CAREFUL-PAINT attribute....1426
CASE-SENSITIVE attribute....1426
CENTERED attribute....1426
CHARSET attribute....1427

CHECKED attribute....1427
CHILD-BUFFER attribute....1427
CHILD-NUM attribute....1427
CLASS-TYPE attribute....1428
CLEAR( ) method (Call Object Handle)....1428
CLEAR( ) method (DEBUGGER Handle)....1428
CLEAR( ) method (FILL-IN Widget)....1429
CLEAR( ) method (ProDataSet Object Handle)....1429
CLEAR( ) method (Temp-Table Object Handle)....1430
CLEAR-APPL-CONTEXT( ) method....1430
CLEAR-LOG( ) method....1431
CLEAR-SELECTION( ) method....1431
CLEAR-SORT-ARROWS( ) method....1432
CLIENT-CONNECTION-ID attribute....1432
CLIENT-TTY attribute....1433
CLIENT-TYPE attribute....1433
CLIENT-WORKSTATION attribute....1433
CLONE-NODE( ) method....1434
CLOSE-LOG( ) method....1434
CODE attribute....1435
CODEPAGE attribute....1435
COLUMN attribute....1435
COLUMN-BGCOLOR attribute....1436
COLUMN-DCOLOR attribute....1436
COLUMN-FGCOLOR attribute....1436
COLUMN-FONT attribute....1437
COLUMN-LABEL attribute....1437
COLUMN-MOVABLE attribute....1437
COLUMN-PFCOLOR attribute....1438
COLUMN-READ-ONLY attribute....1438
COLUMN-RESIZABLE attribute....1438
COLUMN-SCROLLING attribute....1439
COM-HANDLE attribute....1439
COMPLETE attribute....1439
CONFIG-NAME attribute....1440
CONNECT( ) method (AppServer)....1440
CONNECT( ) method (Progress Application Server for OpenEdge)....1446
CONNECT( ) method (Socket object)....1449
CONNECT( ) method (Web service)....1452
CONNECTED( ) method....1457
CONTEXT-HELP attribute....1457
CONTEXT-HELP-FILE attribute....1458
CONTEXT-HELP-ID attribute....1458
CONTROL-BOX attribute....1458
Control-Name property....1459

Controls property....1459
CONVERT-3D-COLORS attribute....1460
CONVERT-TO-OFFSET( ) method....1460
COPY-DATASET( ) method....1461
COPY-SAX-ATTRIBUTES( ) method....1463
COPY-TEMP-TABLE( ) method....1464
COVERAGE attribute....1466
CPCASE attribute....1467
CPCOLL attribute....1467
CPINTERNAL attribute....1467
CPLOG attribute....1468
CPPRINT attribute....1468
CPRCODEIN attribute....1468
CPRCODEOUT attribute....1468
CPSTREAM attribute....1468
CPTERM attribute....1469
CRC-VALUE attribute....1469
CREATE-LIKE( ) method....1470
CREATE-LIKE-SEQUENTIAL( ) method....1471
CREATE-NODE( ) method....1472
CREATE-NODE-NAMESPACE( ) method....1473
CREATE-RESULT-LIST-ENTRY( ) method....1475
CURRENT-CHANGED attribute....1475
CURRENT-COLUMN attribute....1475
CURRENT-ENVIRONMENT attribute....1476
CURRENT-ITERATION attribute (Data Objects)....1476
CURRENT-ITERATION attribute (Widget Objects)....1476
CURRENT-QUERY( ) method....1477
CURRENT-REQUEST-INFO attribute....1479
CURRENT-RESPONSE-INFO attribute....1480
CURRENT-RESULT-ROW attribute....1481
CURRENT-ROW-MODIFIED attribute....1481
CURRENT-WINDOW attribute....1481
CURSOR-CHAR attribute....1482
CURSOR-LINE attribute....1482
CURSOR-OFFSET attribute....1482
DATA-ENTRY-RETURN attribute....1483
DATA-SOURCE attribute....1483
DATA-SOURCE-COMPLETE-MAP attribute....1483
DATA-SOURCE-MODIFIED attribute....1484
DATA-SOURCE-ROWID attribute....1484
DATA-TYPE attribute....1485
DATASET attribute....1485
DATE-FORMAT attribute....1485
DB-CONTEXT attribute....1486

DB-LIST attribute....1486
DB-REFERENCES attribute....1487
DBNAME attribute....1487
DCOLOR attribute....1487
DDE-ERROR attribute....1488
DDE-ID attribute....1489
DDE-ITEM attribute....1489
DDE-NAME attribute....1489
DDE-TOPIC attribute....1490
DEBLANK attribute....1490
DEBUG( ) method....1490
DEBUG-ALERT attribute....1491
DECIMALS attribute....1491
DECLARE-NAMESPACE( ) method....1492
DEFAULT attribute....1492
DEFAULT-BUFFER-HANDLE attribute....1493
DEFAULT-BUTTON attribute....1493
DEFAULT-COMMIT attribute....1494
DEFAULT-STRING attribute....1494
DEFAULT-VALUE attribute....1494
DELETE( ) method....1495
DELETE-CHAR( ) method....1496
DELETE-CURRENT-ROW( ) method....1496
DELETE-HEADER-ENTRY( ) method....1497
DELETE-LINE( ) method....1497
DELETE-NODE( ) method....1497
DELETE-RESULT-LIST-ENTRY( ) method....1498
DELETE-SELECTED-ROW( ) method....1499
DELETE-SELECTED-ROWS( ) method....1499
DELIMITER attribute....1500
DESCRIPTION attribute....1500
DESELECT-FOCUSED-ROW( ) method....1500
DESELECT-ROWS( ) method....1500
DESELECT-SELECTED-ROW( ) method....1501
DETACH-DATA-SOURCE( ) method....1501
DIRECTORY attribute....1501
DISABLE( ) method....1502
DISABLE-AUTO-ZAP attribute....1502
DISABLE-CONNECTIONS( ) method....1502
DISABLE-DUMP-TRIGGERS( ) method....1503
DISABLE-LOAD-TRIGGERS( ) method....1503
DISCONNECT( ) method (Handle)....1504
DISPLAY-MESSAGE( ) method....1504
DISPLAY-TIMEZONE attribute....1505
DISPLAY-TYPE attribute....1505

DOMAIN-DESCRIPTION attribute....1506
DOMAIN-NAME attribute....1506
DOMAIN-TYPE attribute....1507
DOWN attribute....1508
DRAG-ENABLED attribute....1508
DROP-TARGET attribute....1509
DUMP-LOGGING-NOW( ) method....1509
DYNAMIC attribute....1510
EDGE-CHARS attribute....1510
EDGE-PIXELS attribute....1510
EDIT-CAN-PASTE attribute....1510
EDIT-CAN-UNDO attribute....1511
EDIT-CLEAR( ) method....1511
EDIT-COPY( ) method....1511
EDIT-CUT( ) method....1512
EDIT-PASTE( ) method....1512
EDIT-UNDO( ) method....1512
EMPTY attribute....1513
EMPTY-DATASET( ) method....1513
EMPTY-TEMP-TABLE( ) method....1513
ENABLE( ) method....1514
ENABLE-CONNECTIONS( ) method....1514
ENABLE-EVENTS( ) method....1516
ENABLED attribute....1516
ENCODE-DOMAIN-ACCESS-CODE( ) method....1517
ENCODING attribute....1517
ENCRYPT-AUDIT-MAC-KEY( ) method....1520
ENCRYPTION-SALT attribute....1521
END-DOCUMENT( ) method....1521
END-ELEMENT( ) method....1522
END-EVENT-GROUP( ) method....1523
END-FILE-DROP( ) method....1523
END-USER-PROMPT attribute....1524
ENTITY-EXPANSION-LIMIT attribute....1524
ENTRY( ) method....1524
ENTRY-TYPES-LIST attribute....1525
ERROR attribute....1525
ERROR-COLUMN attribute....1526
ERROR-OBJECT attribute....1526
ERROR-OBJECT-DETAIL attribute....1527
ERROR-ROW attribute....1527
ERROR-STACK-TRACE attribute....1528
ERROR-STRING attribute....1528
EVENT-GROUP-ID attribute....1529
EVENT-HANDLER attribute....1529

EVENT-HANDLER-OBJECT attribute....1529
EVENT-PROCEDURE attribute....1530
EVENT-PROCEDURE-CONTEXT attribute....1530
EVENT-TYPE attribute....1531
EXCLUSIVE-ID attribute....1531
EXECUTION-LOG attribute....1531
EXIT-CODE attribute....1531
EXPAND attribute....1532
EXPANDABLE attribute....1533
EXPORT( ) method....1533
EXPORT-PRINCIPAL( ) method....1534
EXTENT attribute....1535
FETCH-SELECTED-ROW( ) method....1535
FGCOLOR attribute....1535
FILE-CREATE-DATE attribute....1536
FILE-CREATE-TIME attribute....1536
FILE-MOD-DATE attribute....1536
FILE-MOD-TIME attribute....1537
FILE-NAME attribute....1537
FILE-OFFSET attribute....1538
FILE-SIZE attribute....1538
FILE-TYPE attribute....1538
FILL( ) method....1539
FILLED attribute....1540
FILL-MODE attribute....1540
FILL-WHERE-STRING attribute....1541
FIND-BY-ROWID( ) method....1541
FIND-CURRENT( ) method....1542
FIND-FIRST( ) method....1543
FIND-LAST( ) method....1545
FIND-UNIQUE( ) method....1546
FIRST-ASYNC-REQUEST attribute....1547
FIRST-BUFFER attribute....1547
FIRST-CHILD attribute....1548
FIRST-COLUMN attribute....1548
FIRST-DATASET attribute....1548
FIRST-DATA-SOURCE attribute....1548
FIRST-FORM attribute....1549
FIRST-OBJECT attribute....1549
FIRST-OF( ) method....1550
FIRST-PROCEDURE attribute....1551
FIRST-QUERY attribute....1551
FIRST-SERVER attribute....1551
FIRST-SERVER-SOCKET attribute....1552
FIRST-SOCKET attribute....1552

FIRST-TAB-ITEM attribute....1552
FIT-LAST-COLUMN attribute....1552
FLAT-BUTTON attribute....1554
FOCUSED-ROW attribute....1554
FOCUSED-ROW-SELECTED attribute....1554
FONT attribute....1554
FOREGROUND attribute....1555
FOREIGN-KEY-HIDDEN attribute....1555
FORM-INPUT attribute....1556
FORM-LONG-INPUT attribute....1556
FORMAT attribute....1556
FORMATTED attribute....1556
FORWARD-ONLY attribute....1557
FRAGMENT attribute....1557
FRAME attribute....1558
FRAME-COL attribute....1558
FRAME-NAME attribute....1558
FRAME-ROW attribute....1558
FRAME-SPACING attribute....1559
FRAME-X attribute....1559
FRAME-Y attribute....1559
FREQUENCY attribute....1560
FULL-HEIGHT-CHARS attribute....1560
FULL-HEIGHT-PIXELS attribute....1560
FULL-PATHNAME attribute....1560
FULL-WIDTH-CHARS attribute....1561
FULL-WIDTH-PIXELS attribute....1561
FUNCTION attribute....1561
GET-ATTRIBUTE( ) method....1561
GET-ATTRIBUTE-NODE( ) method....1562
GET-BINARY-DATA( ) method....1563
GET-BLUE-VALUE( ) method....1563
GET-BROWSE-COLUMN( ) method....1563
GET-BUFFER-HANDLE( ) method....1564
GET-BYTES-AVAILABLE( ) method....1564
GET-CALLBACK-PROC-CONTEXT( ) method....1565
GET-CALLBACK-PROC-NAME( ) method....1565
GET-CGI-LIST( ) method....1566
GET-CGI-VALUE( ) method....1566
GET-CGI-LONG-VALUE( ) method....1566
GET-CHANGES( ) method....1566
GET-CHILD( ) method....1567
GET-CHILD-RELATION( ) method....1568
GET-CLIENT( ) method....1568
GET-COLUMN( ) method....1569

GET-CONFIG-VALUE( ) method....1569
GET-CURRENT( ) method....1570
GET-DATASET-BUFFER( ) method....1570
GET-DOCUMENT-ELEMENT( ) method....1571
GET-DROPPED-FILE( ) method....1571
GET-DYNAMIC( ) method....1572
GET-ERROR-COLUMN( ) method....1572
GET-ERROR-ROW( ) method....1573
GET-FILE-NAME( ) method....1573
GET-FILE-OFFSET( ) method....1573
GET-FIRST( ) method....1574
GET-GREEN-VALUE( ) method....1575
GET-HEADER-ENTRY( ) method....1575
GET-INDEX-BY-NAMESPACE-NAME( ) method....1575
GET-INDEX-BY-QNAME( ) method....1576
GET-ITERATION( ) method (Data Objects)....1576
GET-ITERATION( ) method (Widget Objects)....1577
GET-LAST( ) method....1577
GET-LOCALNAME-BY-INDEX( ) method....1578
GET-MESSAGE( ) method (Handle)....1579
GET-MESSAGE-TYPE( ) method....1579
GET-NEXT( ) method....1580
GET-NODE( ) method....1580
GET-NUMBER( ) method....1581
GET-PARENT( ) method....1581
GET-PREV( ) method....1582
GET-PROPERTY( ) method....1583
GET-PRINTERS( ) method....1583
GET-QNAME-BY-INDEX( ) method....1584
GET-RED-VALUE( ) method....1584
GET-RELATION( ) method....1585
GET-REPOSITIONED-ROW( ) method....1585
GET-RGB-VALUE( ) method....1585
GET-ROW( ) method....1586
GET-SAFE-USER( ) method....1586
GET-SELECTED-WIDGET( ) method....1587
GET-SERIALIZED( ) method....1587
GET-SIGNATURE( ) method....1588
GET-SOCKET-OPTION( ) method....1590
GET-SOURCE-BUFFER( ) method....1591
GET-TAB-ITEM( ) method....1591
GET-TEXT-HEIGHT-CHARS( ) method....1592
GET-TEXT-HEIGHT-PIXELS( ) method....1592
GET-TEXT-WIDTH-CHARS( ) method....1593
GET-TEXT-WIDTH-PIXELS( ) method....1593

GET-TOP-BUFFER( ) method....1594
GET-TYPE-BY-INDEX( ) method....1594
GET-TYPE-BY-NAMESPACE-NAME( ) method....1595
GET-TYPE-BY-QNAME( ) method....1595
GET-URI-BY-INDEX( ) method....1596
GET-VALUE-BY-INDEX( ) method....1596
GET-VALUE-BY-NAMESPACE-NAME( ) method....1597
GET-VALUE-BY-QNAME( ) method....1597
GET-WAIT-STATE( ) method....1597
GRAPHIC-EDGE attribute....1598
GRID-FACTOR-HORIZONTAL attribute....1598
GRID-FACTOR-VERTICAL attribute....1598
GRID-SNAP attribute....1599
GRID-UNIT-HEIGHT-CHARS attribute....1599
GRID-UNIT-HEIGHT-PIXELS attribute....1599
GRID-UNIT-WIDTH-CHARS attribute....1600
GRID-UNIT-WIDTH-PIXELS attribute....1600
GRID-VISIBLE attribute....1600
GROUP-BOX attribute....1601
HANDLE attribute....1601
HANDLER attribute....1602
HAS-LOBS attribute....1603
HAS-RECORDS attribute....1603
Height property....1603
HEIGHT-CHARS attribute....1604
HEIGHT-PIXELS attribute....1604
HELP attribute....1604
HIDDEN attribute....1605
HonorProKeys property....1606
HonorReturnKey property....1606
HORIZONTAL attribute....1607
HTML-CHARSET attribute....1607
HTML-END-OF-LINE attribute....1607
HTML-END-OF-PAGE attribute....1608
HTML-FRAME-BEGIN attribute....1608
HTML-FRAME-END attribute....1608
HTML-HEADER-BEGIN attribute....1608
HTML-HEADER-END attribute....1609
HTML-TITLE-BEGIN attribute....1609
HTML-TITLE-END attribute....1609
HWND attribute....1609
ICFPARAMETER attribute....1610
ICON attribute....1610
IGNORE-CURRENT-MODIFIED attribute....1610
IMAGE attribute....1610

IMAGE-DOWN attribute....1611
IMAGE-INSENSITIVE attribute....1611
IMAGE-UP attribute....1611
IMMEDIATE-DISPLAY attribute....1611
IMPORT-NODE( ) method....1612
IMPORT-PRINCIPAL( ) method....1612
INCREMENT-EXCLUSIVE-ID( ) method....1613
INDEX attribute....1613
INDEX-INFORMATION attribute....1614
INDEX-INFORMATION( ) method....1615
IN-HANDLE attribute....1615
INHERIT-BGCOLOR attribute....1617
INHERIT-FGCOLOR attribute....1617
INITIAL attribute....1618
INITIALIZE( ) method....1619
INITIALIZE-DOCUMENT-TYPE( ) method....1620
INITIATE( ) method....1622
INNER-CHARS attribute....1622
INNER-LINES attribute....1623
INPUT-VALUE attribute....1623
INSERT( ) method....1623
INSERT-ATTRIBUTE( ) method....1624
INSERT-BACKTAB( ) method....1627
INSERT-BEFORE( ) method....1627
INSERT-FILE( ) method....1628
INSERT-ROW( ) method....1628
INSERT-STRING( ) method....1628
INSERT-TAB( ) method....1629
INSTANTIATING-PROCEDURE attribute....1629
INTERNAL-ENTRIES attribute....1631
INVOKE( ) method (Handle)....1631
IS-CLASS attribute....1633
IS-JSON attribute....1633
IS-MULTI-TENANT attribute....1633
IS-OPEN attribute....1633
IS-PARAMETER-SET attribute....1634
IS-PARTITIONED attribute....1634
IS-ROW-SELECTED( ) method....1634
IS-SELECTED( ) method....1635
IS-XML attribute....1635
ITEMS-PER-ROW attribute....1635
KEEP-CONNECTION-OPEN attribute....1636
KEEP-FRAME-Z-ORDER attribute....1636
KEEP-SECURITY-CACHE attribute....1636
KEY attribute....1637

KEYS attribute....1637
LABEL attribute....1638
LABEL-BGCOLOR attribute....1638
LABEL-DCOLOR attribute....1639
LABEL-FGCOLOR attribute....1639
LABEL-FONT attribute....1639
LABELS attribute....1639
LABELS-HAVE-COLONS attribute....1640
LANGUAGES attribute....1640
LARGE attribute....1640
LARGE-TO-SMALL attribute....1640
LAST-ASYNC-REQUEST attribute....1641
LAST-BATCH attribute....1641
LAST-CHILD attribute....1641
LAST-FORM attribute....1642
LAST-OBJECT attribute....1642
LAST-OF( ) method....1643
LAST-PROCEDURE attribute....1644
LAST-SERVER attribute....1644
LAST-SERVER-SOCKET attribute....1644
LAST-SOCKET attribute....1645
LAST-TAB-ITEM attribute....1645
Left property....1645
LENGTH attribute....1646
LIBRARY attribute....1646
LIBRARY-CALLING-CONVENTION attribute....1646
LINE attribute....1647
LIST-ITEM-PAIRS attribute....1647
LIST-ITEMS attribute....1648
LIST-PROPERTY-NAMES( ) method....1648
LISTINGS attribute....1649
LITERAL-QUESTION attribute....1649
LOAD( ) method (Handle)....1650
LoadControls( ) method....1651
LOAD-DOMAINS( ) method....1652
LOAD-ICON( ) method....1653
LOAD-IMAGE( ) method....1654
LOAD-IMAGE-DOWN( ) method....1655
LOAD-IMAGE-INSENSITIVE( ) method....1656
LOAD-IMAGE-UP( ) method....1657
LOAD-MOUSE-POINTER( ) method....1658
LOAD-SMALL-ICON( ) method....1660
LOCAL-HOST attribute....1661
LOCAL-NAME attribute....1662
LOCAL-PORT attribute....1662

LOCAL-VERSION-INFO attribute....1662
LOCATOR-COLUMN-NUMBER attribute....1663
LOCATOR-LINE-NUMBER attribute....1663
LOCATOR-PUBLIC-ID attribute....1663
LOCATOR-SYSTEM-ID attribute....1664
LOCATOR-TYPE attribute....1664
LOCKED attribute....1664
LOCK-REGISTRATION( ) method....1664
LOG-AUDIT-EVENT( ) method....1665
LOG-ENTRY-TYPES attribute....1666
LOG-THRESHOLD attribute....1669
LOGFILE-NAME attribute....1670
LOGGING-LEVEL attribute....1671
LOGIN-EXPIRATION-TIMESTAMP attribute....1672
LOGIN-HOST attribute....1672
LOGIN-STATE attribute....1673
LOGOUT( ) method....1675
LONGCHAR-TO-NODE-VALUE( ) method....1675
LOOKUP( ) method....1676
MANDATORY attribute....1676
MANUAL-HIGHLIGHT attribute....1676
MARK-NEW( ) method....1677
MARK-ROW-STATE( ) method....1677
MAX-BUTTON attribute....1678
MAX-CHARS attribute....1679
MAX-DATA-GUESS attribute....1679
MAX-HEIGHT-CHARS attribute....1680
MAX-HEIGHT-PIXELS attribute....1680
MAXIMUM-LEVEL attribute....1680
MAX-VALUE attribute....1680
MAX-WIDTH-CHARS attribute....1681
MAX-WIDTH-PIXELS attribute....1681
MD5-VALUE attribute....1681
MEMPTR-TO-NODE-VALUE( ) method....1681
MENU-BAR attribute....1682
MENU-KEY attribute....1682
MENU-MOUSE attribute....1682
MERGE-BY-FIELD attribute....1683
MERGE-CHANGES( ) method....1683
MERGE-ROW-CHANGES( ) method....1684
MESSAGE-AREA attribute....1685
MESSAGE-AREA-FONT attribute....1685
MIN-BUTTON attribute....1686
MIN-COLUMN-WIDTH-CHARS attribute....1686
MIN-COLUMN-WIDTH-PIXELS attribute....1687

MIN-HEIGHT-CHARS attribute....1687
MIN-HEIGHT-PIXELS attribute....1687
MIN-SCHEMA-MARSHAL attribute....1687
MIN-VALUE attribute....1688
MIN-WIDTH-CHARS attribute....1688
MIN-WIDTH-PIXELS attribute....1688
MODIFIED attribute....1689
MOUSE-POINTER attribute....1689
MOVABLE attribute....1689
MOVE-AFTER-TAB-ITEM( ) method....1690
MOVE-BEFORE-TAB-ITEM( ) method....1690
MOVE-COLUMN( ) method....1691
MOVE-TO-BOTTOM( ) method....1692
MOVE-TO-EOF( ) method....1692
MOVE-TO-TOP( ) method....1692
MULTI-COMPILE attribute....1693
MULTIPLE attribute....1693
MULTITASKING-INTERVAL attribute....1694
MUST-UNDERSTAND attribute....1695
Name property....1695
NAME attribute....1695
NAMESPACE-PREFIX attribute....1696
NAMESPACE-URI attribute....1697
NEEDS-APPSERVER-PROMPT attribute....1697
NEEDS-PROMPT attribute....1697
NESTED attribute....1697
NEW attribute....1698
NEW-ROW attribute....1698
NEXT-COLUMN attribute....1698
NEXT-ROWID attribute....1698
NEXT-SIBLING attribute....1699
NEXT-TAB-ITEM attribute....1701
NO-CURRENT-VALUE attribute....1701
NO-EMPTY-SPACE attribute....1701
NO-FOCUS attribute....1702
NONAMESPACE-SCHEMA-LOCATION attribute....1703
NO-SCHEMA-MARSHAL attribute....1703
NO-VALIDATE attribute....1703
NODE-VALUE attribute....1704
NODE-VALUE-TO-LONGCHAR( ) method....1704
NODE-VALUE-TO-MEMPTR( ) method....1705
NORMALIZE( ) method....1706
NUM-BUFFERS attribute....1706
NUM-BUTTONS attribute....1706
NUM-CHILD-RELATIONS attribute....1706

NUM-CHILDREN attribute....1707
NUM-COLUMNS attribute....1707
NUM-DROPPED-FILES attribute....1707
NUM-ENTRIES attribute....1707
NUM-FIELDS attribute....1708
NUM-FORMATS attribute....1708
NUM-HEADER-ENTRIES attribute....1708
NUM-ITEMS attribute....1708
NUM-ITERATIONS attribute (data objects)....1709
NUM-ITERATIONS attribute (widget objects)....1709
NUM-LINES attribute....1709
NUM-LOCKED-COLUMNS attribute....1709
NUM-LOG-FILES attribute....1710
NUM-MESSAGES attribute....1710
NUM-PARAMETERS attribute....1711
NUM-REFERENCES attribute....1711
NUM-RELATIONS attribute....1712
NUM-REPLACED attribute....1712
NUM-RESULTS attribute....1712
NUM-SELECTED-ROWS attribute....1712
NUM-SELECTED-WIDGETS attribute....1713
NUM-SOURCE-BUFFERS attribute....1713
NUM-TABS attribute....1713
NUM-TO-RETAIN attribute....1713
NUM-TOP-BUFFERS attribute....1713
NUM-VISIBLE-COLUMNS attribute....1714
NUMERIC-DECIMAL-POINT attribute....1714
NUMERIC-FORMAT attribute....1714
NUMERIC-SEPARATOR attribute....1714
ON-FRAME-BORDER attribute....1715
OPTIONS attribute....1715
ORDINAL attribute....1715
ORIGIN-HANDLE attribute....1716
ORIGIN-ROWID attribute....1716
OVERLAY attribute....1716
OWNER attribute....1717
OWNER-DOCUMENT attribute....1717
PAGE-BOTTOM attribute....1717
PAGE-TOP attribute....1717
PARAMETER attribute....1718
PARENT attribute....1718
PARENT-BUFFER attribute....1719
PARENT-FIELDS-AFTER attribute....1719
PARENT-FIELDS-BEFORE attribute....1719
PARENT-ID-RELATION attribute....1720

PARENT-RELATION attribute....1720
PARSE-STATUS attribute....1720
PASSWORD-FIELD attribute....1721
PATHNAME attribute....1721
PBE-HASH-ALGORITHM attribute....1722
PBE-KEY-ROUNDS attribute....1722
PERSISTENT attribute....1722
PERSISTENT-CACHE-DISABLED attribute....1723
PERSISTENT-PROCEDURE attribute....1723
PFCOLOR attribute....1723
PIXELS-PER-COLUMN attribute....1724
PIXELS-PER-ROW attribute....1724
POPUP-MENU attribute....1724
POPUP-ONLY attribute....1725
POSITION attribute....1725
PREFER-DATASET attribute....1725
PREPARED attribute....1726
PREPARE-STRING attribute....1726
PREV-COLUMN attribute....1726
PREV-SIBLING attribute....1726
PREV-TAB-ITEM attribute....1727
PRIMARY attribute....1728
PRIMARY-PASSPHRASE attribute....1728
PRINTER-CONTROL-HANDLE attribute....1729
PRINTER-HDC attribute....1729
PRINTER-NAME attribute....1730
PRINTER-PORT attribute....1730
PRIVATE-DATA attribute....1730
PROCEDURE-NAME attribute....1731
PROCEDURE-TYPE attribute....1731
PROFILING attribute....1732
PROGRESS-SOURCE attribute....1732
PROXY attribute....1732
PROXY-PASSWORD attribute....1733
PROXY-USERID attribute....1733
PUBLIC-ID attribute....1733
PUBLISHED-EVENTS attribute....1734
QUALIFIED-USER-ID attribute....1734
QUERY attribute....1735
QUERY-CLOSE( ) method....1736
QUERY-OFF-END attribute....1737
QUERY-OPEN( ) method....1737
QUERY-PREPARE( ) method....1738
QUIT attribute....1739
RADIO-BUTTONS attribute....1739

RAW-TRANSFER( ) method....1740
READ( ) method (Socket)....1740
READ-FILE( ) method....1741
READ-JSON( ) method....1742
READ-ONLY attribute....1745
READ-XML( ) method....1746
READ-XMLSCHEMA( ) method....1752
RECID attribute....1757
RECORD-LENGTH attribute....1757
RECURSIVE attribute....1757
REFRESH( ) method (Handle)....1757
REFRESHABLE attribute....1758
REFRESH-AUDIT-POLICY( ) method....1758
REGISTER-DOMAIN( ) method....1759
REJECT-CHANGES( ) method....1760
REJECT-ROW-CHANGES( ) method....1761
REJECTED attribute....1761
RELATION-FIELDS attribute....1761
RELATIONS-ACTIVE attribute....1762
REMOTE attribute....1763
REMOTE-HOST attribute....1763
REMOTE-PORT attribute....1763
REMOVE-ATTRIBUTE( ) method....1764
REMOVE-CHILD( ) method....1765
REMOVE-EVENTS-PROCEDURE( ) method....1765
REMOVE-SUPER-PROCEDURE( ) method....1766
REPLACE( ) method....1767
REPLACE-CHILD( ) method....1769
REPLACE-SELECTION-TEXT( ) method....1770
REPOSITION attribute....1770
REPOSITION-BACKWARD( ) method....1771
REPOSITION-FORWARD( ) method....1771
REPOSITION-TO-ROW( ) method....1772
REPOSITION-TO-ROWID( ) method....1772
REQUEST-INFO attribute....1774
RESET( ) method....1775
RESIZABLE attribute....1776
RESIZE attribute....1776
RESPONSE-INFO attribute....1776
RESTART-ROW attribute....1778
RESTART-ROWID attribute....1779
RETAIN-SHAPE attribute....1780
RETURN-INSERTED attribute....1780
RETURN-VALUE attribute....1780
RETURN-VALUE-DATA-TYPE attribute....1781

RETURN-VALUE-DLL-TYPE attribute....1782
ROLE attribute....1782
ROLES attribute....1783
ROUNDED attribute....1783
ROW attribute....1784
ROW-HEIGHT-CHARS attribute....1784
ROW-HEIGHT-PIXELS attribute....1784
ROW-STATE attribute....1785
ROWID attribute....1785
ROW-MARKERS attribute....1786
ROW-RESIZABLE attribute....1786
SAVE( ) method....1786
SAVE-FILE( ) method....1788
SAVE-ROW-CHANGES( ) method....1789
SAVE-WHERE-STRING attribute....1790
SAX-PARSE( ) method....1790
SAX-PARSE-FIRST( ) method....1791
SAX-PARSE-NEXT( ) method....1792
SCHEMA-CHANGE attribute....1792
SCHEMA-LOCATION attribute....1793
SCHEMA-MARSHAL attribute....1793
SCHEMA-PATH attribute....1794
SCREEN-LINES attribute....1795
SCREEN-VALUE attribute....1795
SCROLL-BARS attribute....1796
SCROLL-TO-CURRENT-ROW( ) method....1796
SCROLL-TO-ITEM( ) method....1796
SCROLL-TO-SELECTED-ROW( ) method....1797
SCROLLABLE attribute....1797
SCROLLBAR-HORIZONTAL attribute....1798
SCROLLBAR-VERTICAL attribute....1798
SEAL( ) method....1798
SEAL-TIMESTAMP attribute....1800
SEARCH( ) method....1800
SELECT-ALL( ) method....1801
SELECT-FOCUSED-ROW( ) method....1802
SELECT-NEXT-ROW( ) method....1803
SELECT-PREV-ROW( ) method....1803
SELECT-ROW( ) method....1803
SELECTABLE attribute....1804
SELECTED attribute....1804
SELECTION-END attribute....1805
SELECTION-START attribute....1805
SELECTION-TEXT attribute....1805
SENSITIVE attribute....1806

SEPARATORS attribute....1806
SEPARATOR-FGCOLOR attribute....1807
SERIALIZE-HIDDEN attribute....1807
SERIALIZE-NAME attribute....1807
SERIALIZE-ROW( ) method....1808
SERVER attribute....1811
SERVER-CONNECTION-BOUND attribute....1812
SERVER-CONNECTION-BOUND-REQUEST attribute....1812
SERVER-CONNECTION-CONTEXT attribute....1813
SERVER-CONNECTION-ID attribute....1813
SERVER-OPERATING-MODE attribute....1814
SESSION-END attribute....1815
SESSION-ID attribute....1815
SET-ACTOR( ) method....1815
SET-APPL-CONTEXT( ) method....1816
SET-ATTRIBUTE( ) method....1817
SET-ATTRIBUTE-NODE( ) method....1818
SET-BLUE-VALUE( ) method....1818
SET-BREAK( ) method....1819
SET-BUFFERS( ) method....1820
SET-CALLBACK( ) method....1821
SET-CALLBACK-PROCEDURE( ) method....1822
SET-CLIENT( ) method....1823
SET-COMMIT( ) method....1826
SET-CONNECT-PROCEDURE( ) method....1826
SET-DYNAMIC( ) method....1827
SET-GREEN-VALUE( ) method....1827
SET-INPUT-SOURCE( ) method....1828
SET-MUST-UNDERSTAND( ) method....1829
SET-NODE( ) method....1829
SET-NUMERIC-FORMAT( ) method....1830
SET-OUTPUT-DESTINATION( ) method....1831
SET-PARAMETER( ) method (Handle)....1832
SET-PROPERTY( ) method....1834
SET-READ-RESPONSE-PROCEDURE( ) method....1835
SET-RED-VALUE( ) method....1836
SET-REPOSITIONED-ROW( ) method....1836
SET-RGB-VALUE( ) method....1837
SET-ROLE( ) method....1837
SET-ROLLBACK( ) method....1838
SET-SAFE-USER( ) method....1838
SET-SELECTION( ) method....1839
SET-SERIALIZED( ) method....1840
SET-SOCKET-OPTION( ) method....1841
SET-SORT-ARROW( ) method....1842

SET-WAIT-STATE( ) method....1844
SHOW-IN-TASKBAR attribute....1844
SIDE-LABEL-HANDLE attribute....1845
SIDE-LABELS attribute....1845
SIGNATURE-VALUE attribute....1846
SINGLE-RUN attribute....1846
SINGLETON attribute....1846
SKIP-DELETED-RECORD attribute....1846
SMALL-ICON attribute....1847
SMALL-TITLE attribute....1847
SOAP-FAULT-ACTOR attribute....1847
SOAP-FAULT-CODE attribute....1847
SOAP-FAULT-DETAIL attribute....1848
SOAP-FAULT-MISUNDERSTOOD-HEADER attribute....1848
SOAP-FAULT-NODE attribute....1848
SOAP-FAULT-ROLE attribute....1848
SOAP-FAULT-STRING attribute....1849
SOAP-FAULT-SUBCODE attribute....1849
SOAP-VERSION attribute....1849
SORT attribute....1850
SORT-ASCENDING attribute....1850
SORT-NUMBER attribute....1850
SSL-SERVER-NAME attribute....1851
STANDALONE attribute....1851
START-DOCUMENT( ) method....1851
START-ELEMENT( ) method....1852
STARTUP-PARAMETERS attribute....1853
STATE-DETAIL attribute....1855
STATISTICS attribute....1856
STATUS-AREA attribute....1856
STATUS-AREA-FONT attribute....1856
STOP attribute....1857
STOP-OBJECT attribute....1857
STOP-PARSING( ) method....1857
STOPPED attribute....1858
STREAM attribute....1858
STRETCH-TO-FIT attribute....1858
STRICT attribute....1859
STRICT-ENTITY-RESOLUTION attribute....1859
STRING-VALUE( ) method....1859
SUBTYPE attribute....1860
SUPER-PROCEDURES attribute....1861
SUPPRESS-NAMESPACE-PROCESSING attribute....1861
SUPPRESS-WARNINGS attribute....1861
SUPPRESS-WARNINGS-LIST attribute....1861

SYMMETRIC-ENCRYPTION-ALGORITHM attribute....1862
SYMMETRIC-ENCRYPTION-IV attribute....1862
SYMMETRIC-ENCRYPTION-KEY attribute....1862
SYMMETRIC-SUPPORT attribute....1863
SYNCHRONIZE( ) method....1864
SYSTEM-ALERT-BOXES attribute....1864
SYSTEM-ID attribute....1865
TAB-POSITION attribute....1865
TAB-STOP attribute....1865
TABLE attribute....1865
TABLE-CRC-LIST attribute....1866
TABLE-HANDLE attribute....1866
TABLE-LIST attribute....1866
TABLE-NUMBER attribute....1867
Tag property (ActiveX control)....1867
TEMP-DIRECTORY attribute....1867
TEMP-TABLE-PREPARE( ) method....1868
TENANT-ID( ) method....1868
TENANT-NAME( ) method....1869
TEXT-SELECTED attribute....1870
THREAD-SAFE attribute....1870
THREE-D attribute....1871
TIC-MARKS attribute....1871
TIME-SOURCE attribute....1872
TIMEZONE attribute....1872
TITLE attribute....1873
TITLE-BGCOLOR attribute....1873
TITLE-DCOLOR attribute....1874
TITLE-FGCOLOR attribute....1874
TITLE-FONT attribute....1874
TOGGLE-BOX attribute....1875
TOOLTIP attribute....1875
TOOLTIPS attribute....1875
Top property....1875
TOP-NAV-QUERY attribute....1876
TOP-ONLY attribute....1876
TRACE-FILTER attribute....1877
TRACING attribute....1878
TRACKING-CHANGES attribute....1878
TRANSACTION attribute....1879
TRANSPARENT attribute....1879
TRANS-INIT-PROCEDURE attribute....1880
TYPE attribute....1880
UNDO attribute....1881
UNDO-THROW-SCOPE attribute....1881

UNIQUE-ID attribute....1882
UNIQUE-MATCH attribute....1882
UPDATE-ATTRIBUTE( ) method....1882
URL attribute....1884
URL-DECODE( ) method....1884
URL-ENCODE( ) method....1884
URL-PASSWORD attribute....1884
URL-USERID attribute....1884
USER-DATA ( ) method....1885
USER-ID attribute....1885
V6DISPLAY attribute....1886
VALIDATE( ) method....1887
VALIDATE-DOMAIN-ACCESS-CODE( ) method....1888
VALIDATE-EXPRESSION attribute....1888
VALIDATE-MESSAGE attribute....1888
VALIDATE-SEAL( ) method....1889
VALIDATE-XML attribute....1890
VALIDATION-ENABLED attribute....1890
VALUE attribute....1890
VERSION attribute....1891
VIEW-AS attribute....1891
VIEW-FIRST-COLUMN-ON-REOPEN attribute....1892
VIRTUAL-HEIGHT-CHARS attribute....1892
VIRTUAL-HEIGHT-PIXELS attribute....1893
VIRTUAL-WIDTH-CHARS attribute....1893
VIRTUAL-WIDTH-PIXELS attribute....1893
VISIBLE attribute....1893
WARNING attribute....1895
WC-ADMIN-APP attribute....1895
WHERE-STRING attribute....1895
Widget-Handle property....1896
WIDGET-ENTER attribute....1897
WIDGET-ID attribute....1897
WIDGET-LEAVE attribute....1898
Width property....1899
WIDTH-CHARS attribute....1899
WIDTH-PIXELS attribute....1900
WINDOW attribute....1900
WINDOW-STATE attribute....1900
WINDOW-SYSTEM attribute....1901
WORD-WRAP attribute....1902
WORK-AREA-HEIGHT-PIXELS attribute....1902
WORK-AREA-WIDTH-PIXELS attribute....1902
WORK-AREA-X attribute....1903
WORK-AREA-Y attribute....1903

WRITE( ) method (Socket)....1903
WRITE-CDATA( ) method....1904
WRITE-CHARACTERS( ) method....1905
WRITE-COMMENT( ) method....1905
WRITE-DATA( ) method....1906
WRITE-DATA-ELEMENT( ) method....1906
WRITE-EMPTY-ELEMENT( ) method....1908
WRITE-ENTITY-REF( ) method....1909
WRITE-EXTERNAL-DTD( ) method....1909
WRITE-FRAGMENT( ) method....1910
WRITE-JSON( ) method....1911
WRITE-MESSAGE( ) method....1917
WRITE-PROCESSING-INSTRUCTION( ) method....1918
WRITE-STATUS attribute....1918
WRITE-XML( ) method....1919
WRITE-XMLSCHEMA( ) method....1925
X attribute....1929
X-DOCUMENT attribute....1930
XCODE-SESSION-KEY attribute....1930
XML-DATA-TYPE attribute....1930
XML-ENTITY-EXPANSION-LIMIT attribute....1931
XML-NODE-NAME attribute....1931
XML-NODE-TYPE attribute....1932
XML-SCHEMA-PATH attribute....1933
XML-STRICT-ENTITY-RESOLUTION attribute....1934
XML-SUPPRESS-NAMESPACE-PROCESSING attribute....1935
Y attribute....1936
YEAR-OFFSET attribute....1936

Handle-based Object Events Reference....1937

Introduction to ABL events....1938
- Event priority....1938
- Apply events....1938
- Triggers and low-level keyboard events....1939

Event tables....1940
- Keyboard events....1940
- Mouse events....1944
- High-level widget events....1946
- Direct manipulation events....1951
- Developer events....1954
- Socket events....1955
- ProDataSet events....1956

Class, Interface, and Enumeration Reference.....1961

Progress.ApplicationServer.AdapterTypes enumeration.....1963
Progress.Data.BindingSource class.....1963
Progress.Data.CancelCreateRowEventArgs class.....1969
Progress.Data.ChildrenAllowEdit class.....1970
Progress.Data.ChildrenAllowNew class.....1970
Progress.Data.ChildrenAllowRemove class.....1971
Progress.Data.ChildrenInputValue class.....1971
Progress.Data.ColumnPropDesc class.....1971
Progress.Data.CreateRowEventArgs class.....1972
Progress.Data.DataType enumeration class.....1973
Progress.Data.DummyRow class.....1974
Progress.Data.InputValue class.....1975
Progress.Data.OffEndEventArgs class.....1976
Progress.Data.SortRequestEventArgs class.....1977
Progress.Data.TableDesc class.....1978
Progress.Database.TempTableInfo class.....1979
Progress.Database.VSTTableId class.....1980
Progress.IO.BinarySerializer class.....1981
Progress.IO.ByteOrder enumeration.....1982
Progress.IO.FileInputStream class.....1982
Progress.IO.FileOutputStream class.....1983
Progress.IO.InputStream class.....1984
Progress.IO.JsonSerializer class.....1985
Progress.IO.MemoryInputStream class.....1986
Progress.IO.MemoryOutputStream class.....1989
Progress.IO.OutputStream class.....1992
Progress.Json.JsonError class.....1992
Progress.Json.JsonParser class.....1993
Progress.Json.JsonParserError class.....1994
Progress.Json.ObjectModel.JsonArray class.....1994
Progress.Json.ObjectModel.JsonConstruct class.....1996
Progress.Json.ObjectModel.JsonDataType class.....1997
Progress.Json.ObjectModel.JsonObject class.....1998
Progress.Json.ObjectModel.ObjectModelParser class.....1999
Progress.Lang.AppError class.....2000
Progress.Lang.Class class.....2002
Progress.Lang.Enum class.....2005
Progress.Lang.Error interface.....2005
Progress.Lang.FlagsEnum class.....2006
Progress.Lang.LockConflict class.....2007
Progress.Lang.Object class.....2008
Progress.Lang.OERequestInfo class.....2009

Progress.Lang.OEVersionInfo class....2010
Progress.Lang.ParameterList class....2011
Progress.Lang.ProError class....2012
Progress.Lang.SoapFaultError class....2013
Progress.Lang.Stop class....2014
Progress.Lang.StopAfter class....2015
Progress.Lang.StopError class....2016
Progress.Lang.SysError class....2017
Progress.Lang.UserInterrupt class....2018
Progress.Reflect.AccessMode enumeration....2019
Progress.Reflect.Constructor class....2020
Progress.Reflect.DataType enumeration....2021
Progress.Reflect.Event class....2022
Progress.Reflect.Flags enumeration....2022
Progress.Reflect.Method class....2024
Progress.Reflect.Parameter class....2025
Progress.Reflect.ParameterMode enumeration....2025
Progress.Reflect.Property class....2026
Progress.Reflect.Variable class....2026
Progress.Security.PAMStatus class....2027
Progress.Util.DateTimeHelper class....2028
Progress.Util.DecimalHelper class....2029
Progress.Util.EnumHelper class....2030
Progress.Util.ResourceHelper class....2031
Progress.Util.StringHelper class....2032
Progress.Util.TypeHelper class....2033
Progress.Web.IWebHandler interface....2034
Progress.Windows.Form class....2034
Progress.Windows.FormProxy class....2038
Progress.Windows.IForm interface....2039
Progress.Windows.MDIChildForm class....2040
Progress.Windows.MDIChildRadForm class....2041
Progress.Windows.MDIChildRadRibbonForm class....2042
Progress.Windows.MDIChildShapedForm class....2043
Progress.Windows.OERadForm class....2043
Progress.Windows.OERadRibbonForm class....2044
Progress.Windows.OEShapedForm class....2044
Progress.Windows.UserControl class....2045
Progress.Windows.WindowContainer class....2048
System.Exception class (OpenEdge-enhanced)....2049

Class Properties and Methods Reference....2051

Syntax for describing class method parameter and property data types....2052
AccessMode property....2052

AccessDenied property....2053
AccessRevoked property....2053
AccountDisabled property....2053
AccountExpired property....2053
ActBufferId property....2054
ActIndexId property....2054
ActIOFileId property....2054
ActIOTypeId property....2054
ActOtherId property....2055
ActRecordId property....2055
ActSpaceId property....2055
ActSummaryId property....2056
AdapterType property....2056
Add( ) method (JsonArray)....2056
Add( ) method (JsonObject)....2061
Add( ) method (Progress.Util.EnumHelper)....2063
AddMessage( ) method....2064
AddNull( ) method (JsonArray)....2064
AddNull( ) method (JsonObject)....2065
AddNumber( ) method (JsonArray)....2066
AddNumber( ) method (JsonObject)....2067
AgentId property....2067
AllowEdit property....2068
AllowNew property....2068
AllowRemove property....2068
And( ) method....2069
Append property....2070
ArchiveIndexStatistics property....2070
ArchiveTableStatistics property....2071
AreEqual( ) method....2071
AreNotEqual( ) method....2072
Array property....2073
ArrayIndex property....2073
Ascending property....2073
Assign( ) method (Progress.Data.BindingSource)....2074
AuthenticationAccess property....2074
AuthenticationFailed property....2074
AutoSort property....2074
AutoSync property....2075
AutoUpdate property....2076
AvailableBytes( ) method....2076
BandIndex property....2077
Batching property....2078
BlockId property....2079
Boolean property....2079

BufferHdl property....2079
BufferName property....2079
BuffStatusId property....2080
ByteOrder property....2080
BytesWritten property....2080
CallStack property....2080
CanRead property....2082
CanWrite property....2082
ChildAllowEdit property....2083
ChildAllowNew property....2083
ChildAllowRemove property....2084
ChildInputValue property....2084
ClientContextId property....2085
Clear( ) method (Class)....2087
Clear( ) method (MemoryOutputStream)....2087
Clone( ) method (JsonArray)....2088
Clone( ) method (JsonObject)....2088
Clone( ) method (Progress.Lang.Object)....2089
Close( ) method (IO)....2089
Closed property....2090
CompareTo( ) method....2090
Complement( ) method....2091
Count property....2092
Create( ) method (CHARACTER)....2093
Create( ) method (DATETIME)....2094
Create( ) method (DECIMAL)....2097
Created property....2099
Custom property....2099
Data property....2099
DataType property....2100
DataTypeName property (Reflect)....2100
DbStatusId property....2100
DeclaringClass property....2101
Deserialize( ) method (BinarySerializer)....2101
Deserialize( ) method (JsonSerializer)....2102
Device property....2103
Dispose( ) method....2104
DisposeDialogOnClose property....2104
EmbeddedWindow property....2105
END_OF_ARRAY property....2109
Equals( ) method....2109
Extent property....2110
FieldIndex property....2110
FieldName property....2110
FileName property....2111

FileListId property....2111
Flush( ) method (FileOutputStream)....2111
Flush( ) method (OutputStream)....2112
Formatted property....2112
Get( ) method....2112
GetCharacter( ) method (JsonArray)....2113
GetCharacter( ) method (JsonObject)....2114
GetClass( ) method (Progress.Lang.Class)....2115
GetClass( ) method (Progress.Lang.Object)....2115
GetClientPrincipal( ) method (Progress.Lang.OERequestInfo)....2116
GetCOMHandle( ) method (JsonArray)....2117
GetCOMHandle( ) method (JsonObject)....2117
GetConstructor( ) method....2118
GetConstructors( ) method....2119
GetDate( ) method (JsonArray)....2120
GetDate( ) method (JsonObject)....2120
GetDatetime( ) method (JsonArray)....2121
GetDatetime( ) method (JsonObject)....2122
GetDatetimeTZ( ) method (JsonArray)....2122
GetDatetimeTZ( ) method (JsonObject)....2123
GetDecimal( ) method (JsonArray)....2124
GetDecimal( ) method (JsonObject)....2125
GetEnum( ) method....2125
GetEnumName( ) method....2127
GetEnumNames( ) method....2127
GetEnumValue( ) method....2128
GetEnumValues( ) method....2128
GetEvent( ) method....2129
GetEvents( ) method....2129
GetHandle( ) method (JsonArray)....2130
GetHandle( ) method (JsonObject)....2131
GetHandlerParameters( ) method....2131
GetIndexInfoByID( ) method....2132
GetIndexStatHistoryHandle( ) method....2133
GetInteger( ) method (JsonArray)....2133
GetInteger( ) method (JsonObject)....2134
GetInterfaces( ) method....2135
GetInt64( ) method (JsonArray)....2135
GetInt64( ) method (JsonObject)....2136
GetJsonArray( ) method (JsonArray)....2136
GetJsonArray( ) method (JsonObject)....2138
GetJsonObject( ) method (JsonArray)....2139
GetJsonObject( ) method (JsonObject)....2140
GetJsonText( ) method (JsonArray)....2142
GetJsonText( ) method (JsonObject)....2143

GetLogical( ) method (JsonArray)....2143
GetLogical( ) method (JsonObject)....2144
GetLongchar( ) method (JsonArray)....2145
GetLongchar( ) method (JsonObject)....2146
GetMemptr( ) method (JsonArray)....2146
GetMemptr( ) method (JsonObject)....2147
GetMessage( ) method (Class)....2148
GetMessageNum( ) method....2148
GetMethod( ) method....2149
GetMethods( ) method....2150
GetNames( ) method (JsonObject)....2151
GetParameters( ) method....2151
GetProperties( ) method....2152
GetProperty( ) method....2152
GetPropertyValue( ) method....2153
GetRaw( ) method (JsonArray)....2155
GetRaw( ) method (JsonObject)....2156
GetRecid( ) method (JsonArray)....2156
GetRecid( ) method (JsonObject)....2157
GetRowid( ) method (JsonArray)....2157
GetRowid( ) method (JsonObject)....2158
GetTableInfoByID( ) method....2159
GetTableInfoByPosition( ) method....2160
GetTableStatHistoryHandle( ) method....2161
GetterAccessMode property....2162
GetType( ) method (JsonArray)....2162
GetType( ) method (JsonObject)....2162
GetType( ) method (TypeHelper)....2163
GetVariable( ) method....2164
GetVariables( ) method....2164
GetValue( ) method....2165
GetVSTHandle( ) method....2166
Handle property....2166
HandleRequest( ) method (IWebHandler)....2167
Has( ) method (JsonObject)....2167
HasStatics( ) method....2168
HasWidgetPool( ) method....2168
IgnoreComments property....2168
IndexStatId property....2169
InputValue property....2169
InvalidConfiguration property....2169
Invoke( ) method (Class)....2170
Invoke( ) method (Constructor)....2171
Invoke( ) method (Method)....2172
IsA( ) method....2172

IsAbstract( ) method....2173
IsAbstract property....2174
IsEnum( ) method....2174
IsFinal( ) method....2174
IsFinal property....2175
IsFlagsEnum( ) method....2175
IsFlagSet( ) method....2175
IsGeneric( ) method....2177
IsGreater( ) method....2177
IsGreaterOrEqual( ) method....2178
IsIndexed( ) method....2179
IsIndexed property....2179
IsInterface( ) method....2179
IsLess( ) method....2179
IsLessOrEqual( ) method....2180
IsNull( ) method (JsonArray)....2181
IsNull( ) method (JsonObject)....2182
IsOverride property....2182
IsSerializable( ) method....2182
IsStatic property....2183
Length property (MemoryOutputStream)....2183
Length property (JSON)....2183
Load( ) method (Class)....2183
LoginDenied property....2184
LoginLockout property....2184
MapNullToEmptyString property....2185
MaxDataGuess property....2185
MaxTries property....2186
MissingCredentials property....2186
Mode property....2186
MstrBlkId property....2187
Name property (Reflect)....2187
New( ) method....2187
NewRow property....2188
NextForm property....2188
NEXT-SIBLING property....2189
Null property....2190
Number property....2190
NumHandlerParameters property....2190
NumMessages property....2190
NumParameters property (ParameterList)....2191
NumParameters property (Reflect)....2191
Object property....2191
OEClientType property....2191
OEMaintVersion property....2192

OEMajorVersion property....2192
OEMinorVersion property....2192
Offset property....2193
Or( ) method....2193
OriginatingClass property....2194
Package property....2195
Parse( ) method....2196
ParseFile( ) method....2196
PasswordExpired property....2197
PermissionDenied property....2197
Position property (BindingSource)....2197
Position property (MemoryInputStream)....2198
Position property (Parameter)....2198
PrevForm property....2198
PREV-SIBLING property....2199
ProcedureName property....2199
ProWinHandle property....2200
Publish( ) method....2201
Read( ) method (InputStream)....2201
Read( ) method (JsonArray)....2202
Read( ) method (JsonObject)....2203
Read( ) method (MemoryInputStream)....2205
ReadByte( ) method....2206
ReadDouble( ) method....2207
ReadFloat( ) method....2207
ReadInt64( ) method....2208
ReadLong( ) method....2208
ReadLongchar( ) method....2208
ReadShort( ) method....2209
ReadString( ) method....2209
ReadUnsignedLong( ) method....2209
ReadUnsignedShort( ) method....2210
ReadOnly property....2210
Refresh( ) method (Progress.Data.BindingSource)....2210
RefreshAll( ) method....2211
Remove( ) method (JsonArray)....2212
Remove( ) method (JsonObject)....2213
RemoveMessage( ) method....2213
RequestId property....2214
Reset( ) method (MemoryInputStream)....2214
ReturnExtent property....2215
ReturnType property....2215
ReturnTypeName property....2215
ReturnValue property....2216
RowModified property....2216

RowsAdded property....2216
Serialize( ) method (BinarySerializer)....2217
Serialize( ) method (JsonSerializer)....2218
SessionId property (OERequestInfo)....2220
Set( ) method (JsonArray)....2220
Set( ) method (JsonObject)....2222
Set( ) method (Reflect)....2224
SetByteOrder( ) method....2225
SetClientPrincipal( ) method....2226
SetFields( ) method....2227
SetFlag( ) method....2229
SetNull( ) method (JsonArray)....2230
SetNull( ) method (JsonObject)....2231
SetNumber( ) method (JsonArray)....2231
SetNumber( ) method (JsonObject)....2232
SetParameter( ) method (Class)....2233
SetPropertyValue( ) method....2235
SetterAccessMode property....2237
Severity property....2237
SkipBytes( ) method (FileInputStream)....2237
SkipBytes( ) method (InputStream)....2238
SkipBytes( ) method (MemoryInputStream)....2238
SoapFault property....2239
Sorted property....2239
StatBaseId property....2239
String property....2240
Subscribe( ) method....2240
Success property....2240
Subtract( ) method....2240
SuperClass property....2241
SyncToSelectedRow property....2241
TableName property....2242
TableSchema property....2242
TableStatId property....2243
Tag property (Progress.Data.BindingSource)....2243
TempTableCount property....2243
TempTablePeak property....2243
ThreadId property....2244
ToggleFlag( ) method....2244
ToObject( ) method....2245
ToString( ) method (Object)....2246
ToString( ) method (Reflect)....2248
TransId property....2248
TypeName property....2248
UnknownUser property....2249

UnsetFlag( ) method....2249
Unsubscribe( ) method....2250
UseFullFieldNames property....2250
User property....2251
UserIndexStatId property....2251
UserIOId property....2252
UserTableStatId property....2252
VersionInfo property....2252
Write( ) method (MemoryOutputStream)....2253
Write( ) method (JsonArray)....2254
Write( ) method (JsonConstruct)....2256
Write( ) method (JsonObject)....2258
Write( ) method (OutputStream)....2260
WriteByte( ) method....2261
WriteDouble( ) method....2261
WriteFile( ) method (JsonArray)....2262
WriteFile( ) method (JsonConstruct)....2263
WriteFile( ) method (JsonObject)....2263
WriteFloat( ) method....2264
WriteInt64( ) method....2265
WriteLong( ) method....2265
WriteShort( ) method....2265
WriteStream( ) method (JsonArray)....2266
WriteStream( ) method (JsonConstruct)....2267
WriteStream( ) method (JsonObject)....2268
WriteUnsignedLong( ) method....2268
WriteUnsignedShort( ) method....2269
Xor( ) method....2269

Class Events Reference....2271

Class events....2272
- Define class events....2272
- Define event handlers....2273
- Use class events....2274
- Events and event methods reference....2275

AfterFetchNextBatch event....2275
BeforeFetchNextBatch event....2276
CancelCreateRow event....2277
CreateRow event....2277
OffEnd event (.NET)....2278
PositionChanged event....2279
Publish( ) event method....2280
SortRequest event....2282
Subscribe( ) event method....2283

Unsubscribe( ) event method.....2284

Keyword Index.....2287

Index.....2353

Preface

Purpose

This reference describes ABL (Advanced Business Language), which is the OpenEdge programming language for building business applications. It covers all ABL statements, functions, phrases, operators, preprocessor directives, special symbols, widgets, handles, classes, interfaces, enumerations, attributes, methods, properties, and events.

Audience

This reference is intended for programmers who develop applications using ABL and for anyone who needs to read and understand ABL code.

Organization

This reference consists of the following sections:

- A dictionary of ABL statements, functions, phrases, operators, preprocessors, and special symbols.
- A dictionary of ABL widgets.
- A dictionary of ABL handles.
- A dictionary of ABL attributes and methods (for handles).
- A dictionary of ABL handle-based object events.
- A dictionary of ABL classes, interfaces, and enumerations.
- A dictionary of ABL properties and methods (for classes).
- A dictionary of ABL class events and event methods.
- An index of ABL keywords.

Documentation conventions

See Documentation Conventions for an explanation of the terminology, format, and typographical conventions used throughout the OpenEdge content library.

Structure of reference entries

Each ABL element reference description includes some subset of the following information:

- Platform-restriction notations
- A purpose or description of the language element
- Block properties for all block statements
- Data-movement diagrams for all data-handling statements

Platform-restriction notes

Some language elements and features of ABL do not apply to all software platforms—operating systems, user interfaces, and database management systems—that OpenEdge supports. The documentation tries to note each such platform restriction with the language element title. Some language elements apply to SpeedScript programming and some do not; the documentation indicates which language elements do not apply with a note in the language element description. You can consider a language element as supported for all interfaces, on all operating systems, and for SpeedScript unless otherwise indicated in the language element description. The platform restriction notes that appear in the documentation include the following:

- **AppServer only**

  The element or feature applies only to the OpenEdge AppServer.

- **Character interfaces only**

  The element or feature applies only to the character interfaces that OpenEdge supports.

- **Graphical interfaces only**

  The element or feature applies only to the graphical interfaces that OpenEdge supports.

- **NT and UNIX only**

  The element or feature applies only to the Windows and UNIX versions that OpenEdge supports.

- **ORACLE only**

  The element or feature applies only to the ORACLE versions that OpenEdge supports.

- **PAS for OE only**

  The element or feature applies only to the Progress Application Server for OpenEdge.

- **UNIX only**

  The element or feature applies only to the UNIX versions that OpenEdge supports.

- **Windows only**

  The element or feature applies only to the Windows versions that OpenEdge supports.

- **Windows only; Graphical interfaces only**

  The element or feature applies only to the graphical interfaces of the traditional OpenEdge GUI for the Windows versions that OpenEdge supports.

- **Windows only; GUI for .NET only**

  The element or feature applies only to the .NET forms, controls, or other .NET objects for the Windows versions that OpenEdge supports.

For a complete list of the software platforms that OpenEdge supports, see the *OpenEdge Platform & Product Availability Guide*.

Purpose

This reference describes ABL (Advanced Business Language), which is the OpenEdge programming language for building business applications. It covers all ABL statements, functions, phrases, operators, preprocessor directives, special symbols, widgets, handles, classes, interfaces, enumerations, attributes, methods, properties, and events.

Audience

This reference is intended for programmers who develop applications using ABL and for anyone who needs to read and understand ABL code.

Organization

This reference consists of the following sections:

- A dictionary of ABL statements, functions, phrases, operators, preprocessors, and special symbols.
- A dictionary of ABL widgets.
- A dictionary of ABL handles.
- A dictionary of ABL attributes and methods (for handles).
- A dictionary of ABL handle-based object events.
- A dictionary of ABL classes, interfaces, and enumerations.
- A dictionary of ABL properties and methods (for classes).
- A dictionary of ABL class events and event methods.
- An index of ABL keywords.

Documentation conventions

See Documentation Conventions for an explanation of the terminology, format, and typographical conventions used throughout the OpenEdge content library.

Structure of reference entries

Each ABL element reference description includes some subset of the following information:

- Platform-restriction notations
- A purpose or description of the language element
- Block properties for all block statements
- Data-movement diagrams for all data-handling statements

Platform-restriction notes

Some language elements and features of ABL do not apply to all software platforms—operating systems, user interfaces, and database management systems—that OpenEdge supports. The documentation tries to note each such platform restriction with the language element title. Some language elements apply to SpeedScript programming and some do not; the documentation indicates which language elements do not apply with a note in the language element description. You can consider a language element as supported for all interfaces, on all operating systems, and for SpeedScript unless otherwise indicated in the language element description. The platform restriction notes that appear in the documentation include the following:

- **AppServer only**

  The element or feature applies only to the OpenEdge AppServer.

- **Character interfaces only**

  The element or feature applies only to the character interfaces that OpenEdge supports.

- **Graphical interfaces only**

  The element or feature applies only to the graphical interfaces that OpenEdge supports.

- **NT and UNIX only**

  The element or feature applies only to the Windows and UNIX versions that OpenEdge supports.

- **ORACLE only**

  The element or feature applies only to the ORACLE versions that OpenEdge supports.

- **PAS for OE only**

  The element or feature applies only to the Progress Application Server for OpenEdge.

- **UNIX only**

  The element or feature applies only to the UNIX versions that OpenEdge supports.

- **Windows only**

  The element or feature applies only to the Windows versions that OpenEdge supports.

- **Windows only; Graphical interfaces only**

  The element or feature applies only to the graphical interfaces of the traditional OpenEdge GUI for the Windows versions that OpenEdge supports.

- **Windows only; GUI for .NET only**

  The element or feature applies only to the .NET forms, controls, or other .NET objects for the Windows versions that OpenEdge supports.

For a complete list of the software platforms that OpenEdge supports, see the *OpenEdge Platform & Product Availability Guide*.

2

Introduction to the ABL Reference

This document describes ABL (Advanced Business Language), which is the OpenEdge programming language for building business applications.

It contains the following sections:

- ABL Syntax Reference
- Widget Reference
- Handle Reference
- Handle Attributes and Methods Reference
- Handle-based Object Events Reference
- Class, Interface, and Enumeration Reference
- Class Properties and Methods Reference
- Class Events Reference
- Keyword Index

3

ABL Syntax Reference

This section contains reference entries that describe the syntax of ABL. They begin with descriptions of the language punctuation and special characters. The remaining entries contain descriptions of the ABL statements, functions, phrases, preprocessor directives, and miscellaneous other language elements.

You can consider a language element as supported for all interfaces, on all operating systems, and for SpeedScript unless otherwise indicated in each language element description. These descriptions refer to both compile-time and run-time behavior, features that the language generally supports and determines at compile time and actions directed by using these features at run time. When describing compile-time features or actions, this section references ABL or the ABL compiler. When describing ABL-directed actions taken at run time, this section references the ABL Virtual Machine (AVM).

: Punctuation

The colon (:) symbol ends block labels and block header statements like DO, FOR, and REPEAT. It also serves as a separator between:

- A handle reference and an attribute or method, for example `ttCust:PRIVATE-DATA` or `ttCust:CLEAR( )`, where `ttCust` is a handle to a temp-table
- An object or static type-name reference and a class-based property, method, or event, for example, `rObj:ToString( )`, `rObj:NEXT-SIBLING`, or `Progress.Lang.Class:GetClass( "Progress.Data.BindingSource" )`, where `rObj` is an object reference to a `Progress.Lang.Object` instance
- A character string literal and one of its options, for example, `"City/State/Zip":U`

It also follows the EDITING keyword in an EDITING phrase and is a device delimiter in Windows.

See :: Punctuation for rules to help you decide whether to use period, colon, or double colon syntax.

See also

: Punctuation, : Punctuation, " "Character-string literal, Class-based object reference, Type-name syntax

:: Punctuation

The double colon (::) is a short-hand syntax for referencing constant named members of database object containers, for example dynamic ProDataSets, queries, temp-tables, and buffers.

The following rules will help you to decide whether to use dot, colon, or double colon syntax:

- Use a dot between two names when the left-hand side name is the actual name of a database or table, known at compile time, and is not a handle or reference of any kind. For example, `Customer.CustNum` or `Sports2020.Customer` or `Sports2020.Customer.CustNum`.
- Use a colon between two character strings when the left-hand side is a handle or reference, and is not the actual name of a database or table, and the right-hand side is an attribute or method for the left-hand-side handle or reference. For example, `hBuff:NUM-FIELDS`, or `hDset:NUM-BUFFERS` or `hBuff:FIND-FIRST`.
- Use a double colon between two character strings when the left-hand side name is a handle to a container object of some kind (buffer, table or ProDataSet) and the right-hand side is not an attribute or method, but instead is a named member of the left-hand side. For example, `hBuff::CustNum` or `hDset::Customer` or `hDSet::Customer::CustNum`.

For example, if `hBuff` is a HANDLE to the `Customer` table, then `hBuff:NAME` returns the string "Customer", but `hBuff::NAME` returns the value of the `NAME` field for the current record in the `hBuff` Buffer, e.g. `"Lift Line Skiing"`.

See also

: Punctuation, . Punctuation

. Punctuation

The period (.) symbol ends all statements, including block header statements. The DO, FOR, and REPEAT statements can end with a period or a colon.

It is also serves as a separator between:

- A filename and a filename extension (suffix) in most operating system platforms, for example, `Letter.txt`
- The elements of a qualified database table or buffer field name, for example, `Customer.CustNum` or `Sports2020.Customer` or `Sports2020.Customer.CustNum`
- The elements of an ABL package name or .NET namespace, for example, `Progress.Lang`, `Progress.Windows`, or `System.Collections`
- An ABL package or .NET namespace and the name of a class or interface defined in that package or namespace, for example, `Progress.Lang.Error`, `Progress.Windows.Form`, or `System.Collections.SortedList`

See : Punctuation for rules to help you decide whether to use period, colon, or double colon syntax.

See also

: Punctuation, :: Punctuation, Type-name syntax

; Punctuation

The ANSI SQL (-Q) startup parameter allows you to redefine the semicolon as a terminator. This startup parameter enforces strict ANSI SQL conformance and allows you to terminate SQL statements with a semicolon. The ANSI SQL (-Q) parameter allows OpenEdge to run standard SQL statements and scripts built with other products.

The ANSI SQL (-Q) parameter disables the use of the semicolon within UNIX escapes.

```
UNIX SMBL=foo; export SMBL
```

As a general rule, use the period (.) as a terminator for ABL statements even when you specify the ANSI SQL (-Q) parameter for an ABL session.

, Punctuation

The comma (,) symbol separates multiple file specifications (used in FOR statements, FOR phrases of DO and REPEAT statements, and PRESELECT phrases), branching statements (used in UNDO statements and phrases), and multiple arguments of a function.

; Special character

This special character is supported only for backward compatibility.

The semicolon (;), when combined with a second character in the Procedure Editor, provides alternative representations of special characters as follows:

Special Character	@	[	]	^	'	{	|	}	~
Alternative Representation	;&	;<	;>	;*	;'	;(	;%	;)	;?

To suppress the semicolon's interpretation as a special character, precede it with a tilde (~). For example, to enter the string **;<** in the Procedure Editor and not have ABL interpret it as an open bracket, type **~;<.**

Additionally, if an ASCII character is mapped to an extended alphabetical character by an IN statement in the PROTERMCAP file, you can enter the extended character in the Procedure Editor by preceding the ASCII character with a semicolon. For example, if [is mapped to Ä, ABL interprets the **;[** sequence as Ä.

? Special character

The question mark is a special character that represents the Unknown value. ABL treats a quoted question mark ("?") in a procedure or an input field as a question mark character. It treats an unquoted question mark (?) in a procedure or an input field as an unknown value.

The following table indicates the results when using the Unknown value (?) in a comparison expression (EQ, GE, GT, LE, LT, NE). These results are true for both character and integer variables.

Table 1: Using the Unknown value (?) in comparison operations

Comparison operator	One argument is ?	Both arguments are ?
EQ or =	F	T
GE or >=	?	T
GT or >	?	F
LE or <=	?	T
LT or <	?	F
NE or <>	T	F

Note: WebSpeed treats an unquoted question mark (?) in an HTML input field as a character.

Additional points about the Unknown value (?) are:

- Any number of Unknown value (?) records can be in a unique index. This is useful in cases where you want to defer choosing key values for a unique index.
- If you define a field as mandatory in the Dictionary, that field cannot contain the Unknown value (?) when the AVM writes the record to the database.
- For sorting and indexing purposes, the Unknown value (?) sorts high.
- The question mark (?) character in the first position of a field equals the Unknown value (?), not a question mark.
- When using the Unknown value (?) in a comparison expression for SQL, the result is unknown.
- When using the Unknown value (?) in an expression, the result of that expression is usually unknown. For example, when you concatenate first, middle, and last names, and the middle name is ?, then the result is ?.

For information on how the Unknown value (?) works with logical data types, comparison operators, and conditional statements, see the following reference entries: EQ or = operator, GE or >= operator, GT or > operator, IF...THEN...ELSE statement, LE or < = operator, LT or < operator, NE or <> operator

\ Special character

The backslash (\) is an escape character on UNIX platforms only. It is a directory path separator in Windows platforms only.

~ Special character

The tilde (~) is an escape character that causes the AVM to read the following character literally. A tilde followed by three octal digits represents a single character. Use it as a lead-in to enter the special characters shown in the following table. In a procedure, a tilde followed by something other than the items in the following table is ignored. For example, `"~abc"` is treated as `"abc"`. (This may not work as expected when passing parameters to an include file.) The items in the following table are case sensitive.

Table 2: Entering special characters in the Procedure Editor

Sequence	Interpreted as	Comment
~"	"	Use within quoted strings as an alternative to two quotes ("").
~'	'	Use within quoted strings as an alternative to two apostrophes (').
~~	~	–
~\	\	–
~{	{	–
~*nnn*	A single character	Where *nnn* is an octal value between 000 and 377. All three digits are required. Do not use this to represent non-ASCII characters if you are using different codepages for compiling and runtime, or have different codepages between different runtime clients. Instead use the ~u or ~U sequences.
~t	Tab character	Octal 011
~r	Carriage return	Octal 015
~n	New line / Line feed	Octal 012
~E	Escape	Octal 033
~b	Backspace	Octal 010
~f	Form feed	Octal 014
~u*nnnn*	A single character	Where *nnnn* is a hexadecimal value representing a Unicode character in the Basic Multilingual Plane. All four digits are required.
~U*nnnnnn*	A single character	Where *nnnnnn* is a hexadecimal value representing a Unicode character beyond the Basic Multilingual Plane. All six digits are required.

" Special character

The double quote (") encloses character constants or strings. To use quotes within a quoted character string, you must use two double quotes (""), which compile to a single double quote ("), or you must put a tilde (~) in front of any quotes within the quoted character string. (This does not work when passing parameters to an include file.)

See also

" "Character-string literal

' Special character

The function of the single quote (') is the same as the double quote. But, if you use single and double quotes in a statement, the compiler checks the outermost quotes first, giving them precedence over the innermost quotes. For example, `DISPLAY '"test"'` shows up as `"test"`, (ABL reads the double quotes literally), and `DISPLAY "'test2'"` shows up as `'test2'`.

See also

" "Character-string literal

/ Special character

The slash (/) symbol is a directory path separator (UNIX). It is also used for date fields (99/99/99).

See also

" "Character-string literal

( ) Expression precedence

Parentheses raise expression precedence. Also, some functions require you to enclose arguments in parentheses.

See also

/ Division operator, Expression

[ ] Array reference

Square brackets ([ ]) enclose array subscripts ([1], [2], etc.) or ranges (such as, [1 FOR 4]). In a range, you can use a variable for the first element, but the second element must be a constant. The specification [1 FOR 4] causes ABL to start with the first array element and to work with that and the next three elements. Square brackets are also used when specifying initial values for an array. For example, if you define an array variable of extent 3, you might specify initial values as INITIAL [0, 1, 2].

= Special character

See the EQ or = operator, Assignment (=) statement.

< Special character

See the LT or < operator.

< = Special character

See the LE or < = operator.

< > Special character

See the NE or <> operator.

> Special character

See the GT or > operator.

> = Special character

See the GE or >= operator.

" "Character-string literal

Specifies a literal character-string value.

Syntax

```
"characters" [ : [ R | L | C | T ][ U ] [ max-length ] ]
```

characters

The literal contents of the character string.

R | L | C | T

Specifies the justification of the string within its maximum length: right, left, centered, or trimmed, respectively. The default justification depends on how the string is used. If the string is displayed with side labels, the default is right justification. If column labels are used, the defaults are left justification for character fields and right justification for numeric fields. Strings used in expressions are trimmed by default.

- R means right justified and padded on the left with spaces: `"Hello":R10 = "  Hello"`.
- L means left justified and padded on the right with spaces: `"Hello":L10 = "Hello  "`.
- C means centered within the string and padded on both the right and left as needed: `"Hello":C10 = "  Hello  "`.
- T means trimmed of leading and trailing blanks (although storage space and screen space is still allocated for the maximum number of characters): `"  Hello":T10 = "Hello"` (but screen and storage space is still reserved for 10 characters).

U

Specifies that the string is untranslatable. If you do not specify U, then the string is assumed to be translatable.

max-length

The number of characters reserved for the string contents in the text segment. The default is the length of the string itself. You might want to specify a longer length if you expect a translation of the string to be longer. The longest length you can specify is 5120 characters.

Note

If you include the colon (:) after the quoted string, you must supply at least one option. Otherwise, ABL treats the colon as a statement separator.

{ } Argument reference

References the value of an argument that a procedure passes to a called external procedure file or to an include file.

ABL converts each argument to a character format. This conversion removes the surrounding double-quotes if the parameter was specified as a character string constant in the RUN statement or include file reference.

When one procedure is called from another and arguments are used, ABL recompiles the called procedure, substituting the arguments that the calling procedure passes, and then runs the called procedure.

Syntax

```
{ { n | &argument-name } }
```

Enter the braces ({}) as shown; they do not represent syntax notation in this description.

n

The number of the argument being referred to. If *n* = 0, ABL substitutes the name of the current procedure (the name you used when you called it, not the full pathname) as the argument. If *n* = *, ABL substitutes all arguments that the calling procedure passes (but not the name {0}). If you refer to the *n*th parameter and the calling procedure does not supply it, {*n*} is ignored.

&*argument-name*

The name of the argument being referred to. If you refer to an *argument-name* and the calling procedure does not supply it, ABL ignores {&*argument-name*}.

If *argument-name* is an asterisk (*), ABL substitutes all arguments that the calling procedure passes. It also adds quotation marks to each parameter, so you can pass the named argument list through multiple levels of include files.

Note: It is invalid to pass both numbered and named arguments within a single pair of braces. Although this will not cause a compile-time or run-time error, the arguments will not be passed correctly.

Examples

The procedure `r-arg.p` runs procedure `r-arg2.p`, passing the arguments customer and name to `r-arg2.p`. ABL substitutes these arguments for {1} and {2} in the `r-arg2.p` procedure.

r-arg.p

```
RUN r-arg2.p "customer" "name"
```

r-arg2.p

```
FOR EACH {1}:
  DISPLAY {2}.
END.
```

The `r-inc.p` procedure defines the variables `txt` and `num`, and assigns the values "Progress VERSION" and "7" to them. The `r-inc.p` procedure includes the `r-inc.i`file and passes the `&int` and `&str` arguments to the include file. Because the parameters are named, their order is unimportant. The called procedure can find each argument, regardless of placement. The `r-inc.i` include file displays a message that consists of the passed arguments. The asterisk argument displays all the parameters as they are listed in the `r-inc.p` procedure.

r-inc.p

```
DEFINE VARIABLE cTxt AS CHARACTER NO-UNDO.
DEFINE VARIABLE iNum AS INTEGER   NO-UNDO.

ASSIGN
  cTxt = "Progress VERSION"
  iNum = 12.

{r-inc.i &int=iNum &str=cTxt}
```

r-inc.i

```
MESSAGE {&str}     /* the &str named argument */
        {&int}.    /* the &int named argument */

MESSAGE "An asterisk displays all the arguments:"
        {*}        /* all the arguments passed by the calling procedure */
```

Notes

- If you pass {} arguments using the RUN statement, you cannot precompile the called procedure. When ABL compiles a procedure, it must have all the values the procedure needs. So, if you pass arguments to a procedure you are calling with the RUN statement, the AVM evaluates those arguments when the calling procedure is run, not when it is compiled.
- You can use the name of an include file as an argument to another include file. For example, a reference to {{1}} in an included procedure causes ABL to include the statements from the file with the name that passed as the first argument.
- Use DEFINE PARAMETER to define a run-time parameter in a called subprocedure. Each parameter requires its own DEFINE statement. The parameters must be specified in the RUN statement in the same order as defined with DEFINE statements.
- ABL disregards an empty pair of braces ({}).
- The maximum length of the arguments you can pass to an include file is determined by the Input Characters (`-inp`) startup parameter.
- An argument *`argument-name`* behaves like a scoped preprocessor name. Thus, if you define a preprocessor name, *`argument-name`*, its value replaces the value of any argument *`argument-name`* passed to the same file at the point where the preprocessor name, *`argument-name`*, is defined.

See also

; Special character, { } Include file reference, { } Preprocessor name reference, COMPILE statement, DEFINE PARAMETER statement, RUN statement

---

{ } Include file reference

Causes ABL to retrieve the statements in a file and compile them as part of the main procedure if it encounters the file's name inside braces ({}) when compiling the procedure. You can name arguments you want substituted in the file before compilation.

Syntax

```
{ include-file
[ argument ... | {&argument-name = "argument-value" } ... ] }
```

Enter the braces ({}) as shown; they do not represent syntax notation in this description.

include-file

The name of an external operating system file that contains statements you want included during the compilation of a main procedure. If the file you name has an unqualified pathname, ABL searches directories based on the PROPATH environment variable. This filename follows normal operating system naming conventions and is case sensitive on UNIX. In Windows, the pathname cannot include characters outside of the non-Unicode code page. See *Internationalize ABL Applications* for more information about Unicode and code pages.

When ABL compiles the main procedure (the procedure containing the `{ }` include file reference), it copies the contents of *include-file* into that procedure, substituting any arguments. So, you can use included procedures with arguments even when you precompile a procedure.

argument

A value used by *include-file*, as a *positional argument*.

With positional arguments, the first *argument* replaces `{1}` in the included file, the second *argument* replaces `{2}`, and so on.

&argument-name = "argument-value"

A name/value pair used by *include-file* as a *named argument*. The *argument-name* is the name of the argument you want to pass to the include file. You can use variable names, field names, and reserved words as argument names. The *argument-value* is the value of the argument you pass to the include file. Enclose the *argument-value* in quotation marks, as shown.

With named arguments, *argument-value* replaces *{&argument-name}* in the included file.

Examples

The `r-inc1.p` procedure uses externally defined and maintained files (`r-fcust.i` and `r-dcust.i`) for the layout and display of a customer report. You can use these same include files in many procedures.

r-inc1.p

```
FOR EACH Customer NO-LOCK:
  {r-fcust.i}
  {r-dcust.i}
END.
```

r-fcust.i

```
FORM Customer.CustNum Customer.Name LABEL "Customer Name"
  Customer.Phone FORMAT "999-999-9999".
```

r-dcust.i

```
DISPLAY Customer.CustNum Customer.Name Customer.Phone.
```

The `r-incl2.p` example references an include file (`r-show.i`) that can take up to five arguments, and the main routine passes four arguments.

r-incl2.p

```
DEFINE VARIABLE var1 AS INTEGER NO-UNDO INITIAL 9.
DEFINE VARIABLE var2 AS DECIMAL NO-UNDO INITIAL 6.43.
DEFINE VARIABLE var3 AS LOGICAL NO-UNDO INITIAL TRUE.

/* any statements */
{r-show.i point-A var1 var2 var3}
/* any statements */
```

r-show.i

```
MESSAGE "At" "{1}" "{2}" {2} "{3}" {3} "{4}" {4} "{5}" {5}.
```

When the main procedure is compiled, the line referencing the `r-show.i` include file is replaced by the following line:

```
MESSAGE At point-A var1 9 var2 6.43 var3 yes
```

This example shows how you can use include files to extend ABL. The main procedure uses a new statement, `r-show.i`, to display the values of fields or variables at various points in a procedure. The include file in this example can handle up to five passed arguments. The main procedure only passes four (point-A, var1, var2, and var3).

The `r-custin.p` procedure displays a frame for each customer that you can update with customer information. The procedure includes `r-cstord.i` and passes the named argument `&frame-options`, and the value of the argument (CENTERED ROW 3 NO-LABEL) to the include file. When the include file references the `&frame-options` argument, it uses the value of the argument, and therefore displays the OVERLAY frame `cust-ord` as a centered frame at row 3 without a label.

r-custin.p

```
FOR EACH Customer:
  {r-cstord.i &frame-options = "CENTERED ROW 3 NO-LABEL"}.
  UPDATE Customer.CustNum Customer.Name Customer.Address Customer.Address2
    Customer.City Customer.State Customer.PostalCode Customer.Phone
    Customer.CreditLimit WITH FRAME cust-ord.
END.
```

r-cstord.i

```
FORM "Cust #" AT 1 Customer.CustNum AT 10 SKIP(1)
  Customer.Name AT 10
  customer.Address AT 10
  customer.Address2 AT 10
  customer.City AT 10 Customer.State Customer.PostalCode SKIP(1)
  "Phone " AT 1 Customer.Phone FORMAT "999/999-9999" AT 10
  "Max Crd" AT 1 Customer.CreditLimit AT 10
  WITH FRAME cust-ord OVERLAY {&frame-options}.
```

Include files are particularly useful for using form layouts in multiple procedures, especially if you do not include the keyword FORM or the closing period (.) of the FORM statement. Thus, the following `r-incl3.p` procedure includes the `r-cust.f` file as the definition of a FORM statement.

r-incl3.p

```
FORM {r-cust.f}.
```

r-cust.f

```
Customer.CustNum
Customer.Name SKIP(2)
Customer.State
```

The `r-incl4.p` procedure uses the same include file as a layout for a DISPLAY statement:

r-incl4.p

```
FOR EACH Customer NO-LOCK:
  DISPLAY {r-cust.f} WITH 3 DOWN.
END.
```

Notes

- You can pass a series of positional arguments or a series of named arguments to an include file, but you cannot pass a combination of positional and named arguments to an include file.
- When you use braces to include one procedure in another, ABL does not include the second procedure until it compiles the first one. This technique has the same effect as using the Editor to copy statements into the main procedure. At times, separate include files are easier to maintain.
- You can nest include files. (They can contain references to other include files.) The number of nested include files is limited by the number of file descriptors available on the system.
- When you have a base procedure and want to make several copies of it, changing it slightly each time, use include files with parameters. For example, at times you might only want to change the name of some files or fields used by the procedure.
- If you define a preprocessor name and later pass a compile-time argument with the same name, but a different value, to a procedure or include file, the value of the initial preprocessor name remains unchanged. Thus, a compile-time argument is scoped to the file to which it is passed.

- Instead of maintaining duplicate source files, create a single include file with the variable portions (such as the names of files and fields) replaced by {1}, {2}, etc. Then each procedure you write can use that include file, passing file and field names as arguments.
- You can use the name of an include file as an argument to another include file. For example, a reference to {{1}} in an include file causes ABL to include the statements from the file with the name that passed as the first argument.
- ABL disregards an empty pair of braces ({}).
- If you use double quotes (" ") around arguments in an argument list, ABL removes them. However, if you use single quotes (' '), ABL passes them. To pass one set of double quotes, you must use four sets of double quotes.
- When ABL reads an include file into the source, it appends a space character to the end of an include file. For example, the following include file `r-string.i` contains data that is used by `r-incstr.p`.

  r-string.i

  ```
  abcde
  ```

  r-incstr.p

  ```
  DISPLAY LENGTH("{r-string.i}").
  ```

  Although `r-string.i` contains five letters, when you run `r-incstr.p`, it returns the value 6 because ABL appends a space character to the end of `r-string.i`.
- The maximum length of the arguments you can pass to an include file is determined by the Input Characters (`-inp`) startup parameter.

See also

{ } Argument reference, { } Preprocessor name reference, COMPILE statement, DEFINE PARAMETER statement, RUN statement

---

{ } Preprocessor name reference

References the value of a preprocessor name in any ABL or preprocessor expression.

Syntax

```
{ &preprocessor-name }
```

Enter the braces ({}) as shown; they do not represent syntax notation in this description.

&preprocessor-name

Expands the name, *preprocessor-name*, to its defined value. You can define preprocessor names using either the &GLOBAL-DEFINE preprocessor directive or the &SCOPED-DEFINE preprocessor

directive. ABL also provides a set of built-in preprocessor names that you can reference for a variety of session information. The following table lists each built-in preprocessor name with its description.

Table 3: Built-in preprocessor names

The preprocessor name . . .	Expands to an unquoted string . . .
BATCH-MODE	Equal to "yes" if the Batch (-b) startup parameter was used to start the client session. Otherwise, it expands to "no".
FILE-NAME	That contains the name of the file being compiled.[1] If you want only the name of the file as specified in the { } Include File Reference, the RUN statement, or the COMPILE statement, use the argument reference {0}.
LINE-NUMBER	That contains the current line number in the file being compiled. If you place this reference in an include file, the line number is calculated from the beginning of the include file.
OPSYS	That contains the name of the operating system on which the file is being compiled. The OPSYS name can have the same values as the OPSYS function. The possible values are "UNIX" and "WIN32".[2]
PROCESS-ARCHITECTURE	That contains the bit value of the AVM process on which the file is compiled. The possible values include "32" and "64".[3]
SEQUENCE	Representing a unique integer value that is sequentially generated each time the SEQUENCE preprocessor name is referenced. When a compilation begins, the value of {&SEQUENCE} is 0; each time {&SEQUENCE} is referenced, the value increases by 1. To store the value of a reference to SEQUENCE, you must define another preprocessor name as {&SEQUENCE} at the point in your code you want the value retained.
WINDOW-SYSTEM	That contains the name of the windowing system in which the file is being compiled. The possible values include "MS-WINDOWS", "MS-WIN95", "MS-WINXP", and "TTY".[4]

[1] When running the source code of a procedure file loaded into the Procedure Editor or the AppBuilder, {&FILE-NAME} expands to a temporary filename, not the name of the file under which the source code might be saved.

[2] ABL supports an override option that enables applications that need to return the value of MS-DOS for all Microsoft operating systems to do so. For example, if you do not want the value WIN32 to be returned when the operating system is recognized, you can override this return value by defining the Opsys key in the Startup section of the current environment, which can be in the registry or in an initialization file. If the Opsys key is located, the OPSYS function returns the value associated with the Opsys key on all platforms.

[3] ABL provides only bit value of the AVM process. The PROCESS-ARCHITECTURE preprocessor name lets you generate r-code file for a single target, either for 32- or 64-bit applications.

[4] ABL supports an override option for the &WINDOW-SYSTEM preprocessor name that provides backward compatibility. This option enables applications that need the WINDOW-SYSTEM preprocessor name to return the value of MS-WINDOWS for all Microsoft operating systems to do so. To establish this override value, define the WindowSystem key in the Startup section of the current environment, which can be in the registry or in an initialization file. If the WindowSystem key is located, the WINDOW-SYSTEM preprocessor name returns the value associated with the WindowSystem key on all platforms

Example

This procedure displays a message indicating whether the AVM is running on 32- or 64-bit Windows. This example is not valid if compiled in a 32-bit AVM and run in a 64-bit AVM, and vice versa.

```
DEFINE VARIABLE hProcess AS INTEGER NO-UNDO.
DEFINE VARIABLE Wow64Process AS INTEGER NO-UNDO.
DEFINE VARIABLE resultValue AS INTEGER NO-UNDO.

 &IF {&PROCESS-ARCHITECTURE} = 64 &THEN
  /* 64-bit processes always run on 64-bit Windows */
  MESSAGE "64-bit process running on 64-bit Windows".
&ELSE
  /* If a 32-bit process is running under WOW64 it must be
  ** running on 64-bit Windows. Otherwise, it is running on
  ** 32-bit Windows.
  */
  RUN GetCurrentProcess(OUTPUT hProcess).
  RUN IsWow64Process(hProcess, INPUT-OUTPUT Wow64Process, OUTPUT resultValue).

   MESSAGE "32-bit process running on " + (IF Wow64Process > 0 THEN "64" ELSE "32") +
"-bit Windows".
&ENDIF

 PROCEDURE GetCurrentProcess EXTERNAL "kernel32.dll":
  DEFINE RETURN PARAMETER hProcess AS LONG.
END.

 PROCEDURE IsWow64Process EXTERNAL "kernel32.dll":
  DEFINE INPUT PARAMETER hProcess AS LONG.
  DEFINE INPUT-OUTPUT PARAMETER Wow64Process AS HANDLE TO LONG.
  DEFINE RETURN PARAMETER resultValue AS LONG.
END.
```

The following table lists the additional built-in preprocessor names that apply to SpeedScript.

Table 4: SpeedScript built-in preprocessor names

The preprocessor name . . .	Expands to an unquoted string . . .
DISPLAY	DISPLAY {&WEBSTREAM}
OUT	PUT {&WEBSTREAM} UNFORMATTED
OUT-FMT	PUT {&WEBSTREAM} FORMATTED
OUT-LONG	EXPORT {&WEBSTREAM}
WEBSTREAM	STREAM WebStream

Examples

The `r-prprc1.p` procedure shows how you can reference a built-in preprocessor name and include it in a character string.

r-prprc1.p

```
MESSAGE "The current operating system is" "{&OPSYS}."
  VIEW-AS ALERT-BOX.
```

The procedure `r-prprc2.p` shows how to capture the value of a {&SEQUENCE} reference. In this example, {&SEQUENCE} is referenced three times, once each to assign its value to `wvar` (0) and `xvar` (1) at run time. The third reference defines the preprocessor name Last-Value with the value 3. As shown, Last-Value is assigned unchanged to both `yvar` and `zvar`, each of which takes the value 3 at run time.

r-prprc2.p

```
DEFINE VARIABLE wvar AS INTEGER NO-UNDO.
DEFINE VARIABLE xvar AS INTEGER NO-UNDO.
DEFINE VARIABLE yvar AS INTEGER NO-UNDO.
DEFINE VARIABLE zvar AS INTEGER NO-UNDO.

ASSIGN
  wvar = {&SEQUENCE}
  xvar = {&SEQUENCE}.

&GLOBAL-DEFINE Last-Value {&SEQUENCE}

ASSIGN
  yvar = {&Last-Value}
  zvar = {&Last-Value}.

MESSAGE "wvar =" wvar SKIP "xvar =" xvar SKIP
        "yvar =" yvar SKIP "zvar =" zvar VIEW-AS ALERT-BOX.
```

The procedure `r-prprc3.p` shows how preprocessor names override compile-time arguments. In this example, `r-prprc3.p` defines the preprocessor name My-Name as "Daniel". It then passes the compile-time argument My-Name, with the value "David", to the include file `r-prprc3.i`, which in turn defines a preprocessor name My-Name as "Donald".

r-prprc3.p

```
&SCOPED-DEFINE My-Name "Daniel"
{r-prprc3.i &My-Name = "David"}
MESSAGE "My-Name preprocessed in r-prprc3.p is" {&My-Name} + "."
  VIEW-AS ALERT-BOX.
```

r-prprc3.i

```
MESSAGE "My-Name argument in r-prprc3.i is" "{&My-Name}" + "."
  VIEW-AS ALERT-BOX.
&SCOPED-DEFINE My-Name "Donald"
MESSAGE "My-Name preprocessed in r-prprc3.i is" {&My-Name} + "."
  VIEW-AS ALERT-BOX
```

During execution, the first message included by `r-prprc3.i` displays the value of the My-Name argument, "David". The second message included by `r-prprc3.i` displays the value of the following My-Name preprocessor name, defined as "Donald", permanently overriding "David" passed by the My-Name argument. Finally, the message in `r-prprc3.p` displays the value of the My-Name preprocessor name that was initially defined there, "Daniel", because the value from My-Name established in `r-prprc3.i` ("Donald") went out of scope during compilation.

Note also that the reference to the My-Name compile-time argument in `r-prprc3.i` is inside double-quotes, because ABL passes string constant values for compile-time arguments without the surrounding double-quotes.

You can encounter compilation problems mixing preprocessor names with compile-time argument names. The following example, a variation of `r-prprc3.i`, does not compile, even when passed a My-Name argument as an include file. This is because the preprocessor My-Name value overrides the argument My-Name value, as shown:

```
&SCOPED-DEFINE My-Name "Donald"
MESSAGE "My-Name preprocessed in r-prprc3.i is" {&My-Name} + "."
  VIEW-AS ALERT-BOX.
MESSAGE "My-Name argument in r-prprc3.i is" "{&My-Name}" + "."
  VIEW-AS ALERT-BOX.
```

Because the preprocessor My-Name defines a quoted "Donald" value, ABL replaces "{&My-Name}" in the fourth line with ""Donald"". This appears to the compiler as two empty strings and an unknown variable reference (Donald). Although you can do it with care, in general, avoid using the same names for compile-time arguments and preprocessor names.

Notes

- ABL expands preprocessor names wherever and in whatever context it finds them, including inside quoted character strings.
- If you define a preprocessor name in the same file and with the same name as a compile-time argument passed to the file, the value of the preprocessor name takes precedence over the value of the argument name from the point where the preprocessor name is defined.

See also

&GLOBAL-DEFINE preprocessor directive, &SCOPED-DEFINE preprocessor directive, { } Argument reference, { } Include file reference, ; Special character

---

&GLOBAL-DEFINE preprocessor directive

Globally defines a compile-time constant (preprocessor name).

Syntax

```
&GLOBAL-DEFINE preprocessor-name definition
```

preprocessor-name

The preprocessor name (compile-time constant) that you supply. ABL reserved keywords are allowed, but cannot be used in preprocessor expressions.

definition

A string of characters (or preprocessor references that evaluate to a string of characters) whose content the preprocessor substitutes for *preprocessor-name* during compilation. If the definition is longer than one line, a tilde (~) at the end of a line indicates continuation to the next line.

Examples

In this example, the preprocessor name MAX-EXPENSE is defined as the text string "5000":

```
&GLOBAL-DEFINE MAX-EXPENSE 5000
```

Wherever the reference {&MAX-EXPENSE} appears in the source code, the preprocessor substitutes the text string "5000". For example, the preprocessor changes this line of code:

```
IF tot-amount <= {&MAX-EXPENSE} THEN DO:
```

To this line:

```
IF tot-amount <= 5000 THEN DO:
```

Notes

- You must place the &GLOBAL-DEFINE directive at the beginning of a line, preceded only by blanks, tab characters, or comments (*/* comment */*). The preprocessor trims all leading and trailing spaces from *definition*.
- The syntax of the &GLOBAL-DEFINE and &SCOPED-DEFINE directives are identical but these directives are used differently.
- If duplicate &GLOBAL-DEFINE directives are specified for the same *preprocessor-name*, the last &GLOBAL-DEFINE directive is applied.

See also

{ } Preprocessor name reference, &SCOPED-DEFINE preprocessor directive, &UNDEFINE preprocessor directive, DEFINED preprocessor function

&IF, &THEN, &ELSEIF, &ELSE, and &ENDIF preprocessor directives

These directives set logical conditions for the inclusion of blocks of code to compile.

Syntax

```
&IF expression &THEN
  block
[ &ELSEIF expression &THEN
  block ]...
[ &ELSE
  block ]
&ENDIF
```

expression

An expression that can contain preprocessor name references, the operators listed in Table 6: Preprocessor operators, the ABL functions listed in Table 7: Functions allowed in preprocessor expressions, and the DEFINED( ) preprocessor function.

When it encounters an &IF directive, the preprocessor evaluates the expression that immediately follows. This expression can continue for more than one line; the &THEN directive indicates the end of the expression. If the expression evaluates to TRUE, then the block of code between it and the next &ELSEIF, &ELSE, or &ENDIF is compiled. If the expression evaluates to FALSE, the block of code is not compiled and the preprocessor proceeds to the next &ELSEIF, &ELSE, or &ENDIF directive. No include files referenced in this block of code are included in the final source. You can nest &IF directives.

The expression that follows the &ELSEIF directive is evaluated only if the &IF expression tests false. If the &ELSEIF expression tests TRUE, the block of code between it and the next &ELSEIF, &ELSE, or &ENDIF directive is compiled. If the &ELSEIF expression tests FALSE, the preprocessor proceeds to the next &ELSEIF, &ELSE, or &ENDIF directive.

The block of code between the &ELSE and &ENDIF directives is compiled only if the &IF expression and the &ELSEIF expressions all test false. If there are no &ELSEIF directives, the block of code is compiled if the &IF expression tests false.

Once any &IF or &ELSEIF expression evaluates to TRUE, no other block of code within the &IF...&ENDIF block is compiled.

The &ENDIF directive indicates the end of the conditional tests and the end of the final block of code to compile.

The following table shows how preprocessor expressions are evaluated.

Table 5: Preprocessor expressions

Type of expression	TRUE	FALSE
LOGICAL	TRUE	FALSE
CHARACTER	non-empty	empty
INTEGER	non-zero	0
INT64	non-zero	0
DECIMAL	not supported	not supported

The following table lists the operators supported within preprocessor expressions. These operators have the same precedence as the regular ABL operators.

Table 6: Preprocessor operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
=	Equality
<>	Inequality
>	Greater than
<	Less than
=>	Greater than or equal to
<=	Less than or equal to
AND	Logical and
OR	Logical or
NOT	Logical not
BEGINS	Compares the beginning letters of two expressions
MATCHES	Compares two strings

The following table lists the ABL functions supported within preprocessor expressions.

Table 7: Functions allowed in preprocessor expressions

ABSOLUTE	ASC	AUDIT-ENABLED
DATE	DAY	DBTYPE
DECIMAL	ENCODE	ENTRY
ETIME	EXP	FILL
INDEX	INT64	INTEGER
KEYWORD	KEYWORDALL	LC
LEFT-TRIM	LENGTH	LIBRARY
LOG	LOOKUP	MATCHES

MAXIMUM	MEMBER	MINIMUM
MODULO	MONTH	NUM-ENTRIES
OPSYS	PROCESS-ARCHITECTURE	PROPATH
PROVERSION	RANDOM	REPLACE
RIGHT-TRIM	R-INDEX	ROUND
SQRT	STRING	SUBSTITUTE
SUBSTRING	TIME	TODAY
TRIM	TRUNCATE	WEEKDAY
YEAR		

Note

When the preprocessor evaluates expressions, all arithmetic operations are performed with integers. Preprocessor name references used in arithmetic operations must evaluate to integers.

See also

&GLOBAL-DEFINE preprocessor directive, &SCOPED-DEFINE preprocessor directive, &UNDEFINE preprocessor directive

&MESSAGE preprocessor directive

Displays a message at compile time in the Compiler Messages dialog box.

Note: Does not apply to SpeedScript programming.

Syntax

```
&MESSAGE text-string
```

text-string

A string of characters, preprocessor name references, named include file arguments, or any combination of these that results in a character string to display. The *text-string* argument does not need to be quoted.

Example

This is a possible compile-time message directive:

```
&MESSAGE Compiling the {&FILE-NAME} file.
. . .
```

If this fragment appears in a procedure file, `cmessage.p`, compiling this file with the COMPILE statement causes the following message to be included with the compiler messages:

```
Compiling the cmessage.p file.
```

See also

{ } Preprocessor name reference

&SCOPED-DEFINE preprocessor directive

Defines a compile-time constant (preprocessor name) non-globally.

Syntax

```
&SCOPED-DEFINE preprocessor-name definition
```

preprocessor-name

The preprocessor name (compile-time constant) that you supply. ABL reserved keywords are allowed, but cannot be used in preprocessor expressions.

definition

A string of characters (or preprocessor references that evaluate to a string of characters) whose content the preprocessor substitutes for *preprocessor-name* during compilation. If definition is longer than on line, a tilde (~) at the end of a line indicates continuation to the next line.

Notes

- You must place the &SCOPED-DEFINE directive at the beginning of a line, preceded only by blanks, tab characters, or comments (`/* comment */`). The preprocessor trims all leading and trailing spaces from *definition*.
- The syntax of the &GLOBAL-DEFINE and &SCOPED-DEFINE directives are identical but these directives are used differently.

See also

{ } Preprocessor name reference, &GLOBAL-DEFINE preprocessor directive, &UNDEFINE preprocessor directive, DEFINED preprocessor function

&UNDEFINE preprocessor directive

Undefines a compile-time constant (preprocessor name).

Syntax

```
&UNDEFINE preprocessor-name
```

preprocessor-name

The preprocessor name (compile-time constant) that you want to undefine.

Notes

- When you use the &UNDEFINE directive, ABL warns you if the name you want to undefine was not previously defined.
- The &UNDEFINE directive undefines the currently active name. It also undefines named include file arguments.
- To globally define the same name more than once, use this directive to undefine the name before redefining it. If you do not undefine the global name before redefining it, the compiler produces a warning message for non-globally (scoped) defined names. For globally defined names, &UNDEFINE is optional, but recommended for code readability. If &UNDEFINE is not used, the last &GLOBAL-DEFINE statement is used.

See also

&GLOBAL-DEFINE preprocessor directive, &SCOPED-DEFINE preprocessor directive, DEFINED preprocessor function

/* Multi-line comments */

Allows you to add explanatory text to code. The comment begins with /* and ends with */. This type of comment can wrap over multiple lines.

Syntax

```
/* comment */
```

comment

Descriptive text.

Examples

The `r-comm.p` example uses comments to document the history of procedure modifications.

r-comm.p

```
/* Procedure written  9/5/87 by CHC
            revised  9/27/87 by DG */

FOR EACH Customer NO-LOCK:
  DISPLAY Customer.CustNum Customer.Name Customer.Contact Customer.Phone.
END.
```

The `r-comm2.p` example uses comments to describe what the procedure does.

r-comm2.p

```
/* Step through unshipped orders */
FOR EACH Order NO-LOCK WHERE Order.ShipDate = ?:
  /* Display order date, promise date, terms */
  DISPLAY Order.OrderDate Order.PromiseDate Order.Terms.
  /*
    FOR EACH OrderLine OF Order NO-LOCK:
      /* Display all OrderLines of each Order */
      DISPLAY OrderLine.
    END.
  */
END.
```

The comment symbols that enclose the inner FOR EACH block turn that block into a comment for testing purposes. You can nest comments, so ABL correctly processes any comments already in the bypassed code. Both multi-line comments and single-line comments can be nested inside a multi-line comment.

See also

// Single-line comments

// Single-line comments

Allows you to add explanatory text to code. The comment begins with a double slash (//) and ends at a newline character (\n), a carriage return (\r), or the end of the file.

Syntax

```
// comment
```

comment

Descriptive text.

Note: Comments can be nested.

Examples

This example uses a comment to document a procedure modification:

```
// Procedure written  9/5/87 by CHC
//           revised  9/27/87 by DG

FOR EACH Customer NO-LOCK:
  DISPLAY Customer.CustNum Customer.Name Customer.Contact Customer.Phone.
END.
```

To be properly interpreted as the beginning of a single-line comment, the "//" must be preceded by a line break or white space. In the first line of code below, the text between "//" and the end of the line will be interpreted as a comment, but it will not in the second line:

```
RUN r-upord.p. // Update customer's orders

RUN r-upord.p.// Update customer's orders
```

Caution: If you need to use // at the beginning of a pathname, such as in `RUN //dir1/file`, the pathname will be interpreted as a comment because the // is preceded by a space. Enclosing the pathname in quotes ensures that the text following // will not be treated as a comment. Pathnames that contain // in their interior, such as in `RUN parent//dir1/file`, will work as expected.

Both multi-line comments and single-line comments can be nested inside a single-line comment.

See also

/* Multi-line comments */

+ Unary positive operator

Preserves the positive or negative value of a numeric expression. Do not confuse this operator with the addition operator that you use to add expressions together.

Syntax

```
+ expression
```

expression

An expression whose value is numeric.

Example

In the `r-unpos.p` example, the sign of `CreditLimit` is preserved as is the sign of the sum of `CreditLimit` + 100. The unary positive is not necessary; it is used simply to document the procedure, as shown:

r-unpos.p

```
DEFINE VARIABLE old-max NO-UNDO LIKE CreditLimit LABEL "Old Limit".

FOR EACH Customer:
  old-max =+ Customer.CreditLimit.
  Customer.CreditLimit =+ (Customer.CreditLimit + 100).
  DISPLAY Customer.Name old-max Customer.CreditLimit.
END.
```

See also

- Unary negative operator

+ Addition operator

Adds two numeric expressions.

Syntax

```
expression + expression
```

expression

An expression whose value is numeric.

Example

In the following example, the addition operator (+) adds 100 to the value of the `CreditLimit` field:

r-addn.p

```
FOR EACH Customer:
  Customer.CreditLimit = Customer.CreditLimit + 100.
END.
```

Note

Adding two decimal expressions produces a DECIMAL value. Adding two INTEGER expressions produces an INTEGER value. Adding an integer expression (either INTEGER or INT64) and a decimal expression produces a DECIMAL value. Adding an INTEGER and an INT64 expression produces an INT64 value.

See also

- Subtraction operator

+ Concatenation operator

Joins two character strings or expressions. The data type of the returned value matches the data type of the expressions passed to the function. When one expression is LONGCHAR and the other CHARACTER, the result is a LONGCHAR.

Syntax

```
expression + expression
```

expression

An expression whose value is a character string. If any expression is a LONGCHAR, the result is a LONGCHAR. Also, the result converts to the code page of the expression on the left.

Example

The `r-conc.p` procedure prints mailing labels. It uses the concatenation operator (+) to ensure that the third line of each label shows the city and state separated by a comma and a space. The FORMAT x(16) is specified to provide room for up to 16 characters in the result of the concatenation. If a FORMAT is not given, then the AVM displays only the first eight characters of the result since x(8) is the default format for a character expression.

r-conc.p

```
FOR EACH Customer NO-LOCK:
  DISPLAY SKIP(1) Customer.Name SKIP Customer.Address SKIP
    Customer.City + ", " + Customer.State FORMAT "x(16)" Customer.Country
    Customer.PostalCode SKIP(2).
END.
```

This is a label produced by this procedure:



Notes

If **any** of the string values you concatenate is the Unknown value (?), then the result is the Unknown value (?). This might lead to unexpected results if a field used in an expression is not mandatory. For example, you might have fields for a person's first name, last name, and middle initial. You might combine these into a full name with an expression like the following:

```
DISPLAY fname + " " + minit + " " + lname FORMAT "x(36)".
```

If `minit` is not a mandatory field, it might be set to the Unknown value (?) in some records. If so, then those records are displayed as the Unknown value (?). You can avoid this by using conditional code. For example:

```
DISPLAY fname + " " + (IF minit <> ? THEN minit + ". " ELSE "") + " " +
  lname FORMAT "x(36)".
```

+ Date addition operator

Adds a number of days to a date, producing a date result.

Syntax

```
date + days
```

date

An expression that evaluates to a DATE value.

days

An expression with a value of the number of days you want to add to a *date*.

Example

The `r-dadd.p` procedure finds all unshipped orders that are at least one week overdue. If the order is not shipped and the promised date is more than seven days ago, the procedure finds the record for the customer who placed the order and displays the order and customer data.

r-dadd.p

```
DISPLAY "ORDERS SCHEDULED TO SHIP MORE THAN ONE WEEK LATE".
FOR EACH Order NO-LOCK WHERE Order.ShipDate = ?:
  IF TODAY > (Order.PromiseDate + 7) THEN DO:
    FIND Customer OF Order NO-LOCK.
    DISPLAY Order.OrderNum Order.CustNum Customer.Name Order.PromiseDate
      Customer.Terms.
  END.
END.
```

Notes

- To add a specific number of days and a specific number of milliseconds to a DATETIME, use the DATETIME function. For example:

  ```
  new-datetime = DATETIME( DATE(old-datetime) + days,
                           MTIME(old-datetime) + milliseconds ).
  ```

  The DATETIME function ensures that the time portion remains within the valid range, by adding day(s) to the date part when the time part goes over the number of milliseconds in a day.

- To add a specific number of days and milliseconds to a DATETIME-TZ, use the DATETIME-TZ function. For example:

```
new-datetime-tz = DATETIME-TZ( DATE(old-datetime-tz) + days,
                               MTIME (old-datetime-tz) + milliseconds,
                               TIMEZONE(old-dateime-tz) ).
```

  The DATETIME-TZ function ensures that the time portion remains within the valid range, by adding day(s) to the date portion when the time part goes over the number of milliseconds in a day.

See also

- Date subtraction operator, + Datetime addition operator, ADD-INTERVAL function, DATE function

---

+ Datetime addition operator

Adds a number of milliseconds to a DATETIME or a DATETIME-TZ to produce another DATETIME or DATETIME-TZ.

Syntax

```
datetime + milliseconds
```

```
datetime-tz + milliseconds
```

datetime

An expression that evaluates to a DATETIME value.

milliseconds

An expression that evaluates to an integer value specifying a number of milliseconds.

datetime-tz

An expression that evaluates to a DATETIME-TZ value.

Example

This example returns the date and time exactly 24 hours later (`dtTime`) by calculating and adding the number of milliseconds in 24 hours to the present date and time (NOW function) and displays the result:

```
DEFINE VARIABLE dtTime AS DATETIME NO-UNDO.
DEFINE VARIABLE iHour  AS INTEGER  NO-UNDO INITIAL 3600000.

dtTime = NOW + (24 * iHour).

MESSAGE "A day later: " dtTime VIEW-AS ALERT-BOX.
```

Notes

- To add a specific number of days and a specific number of milliseconds to a DATETIME, use the DATETIME function. For example:

```
new-datetime = DATETIME( DATE(old-datetime) + days,
                         MTIME(old-datetime) + milliseconds ).
```

  The DATETIME function ensures that the time portion remains within the valid range, by adding day(s) to the date part when the time part goes over the number of milliseconds in a day.

- To add a specific number of days and milliseconds to a DATETIME-TZ, use the DATETIME-TZ function. For example:

```
new-datetime-tz = DATETIME-TZ( DATE(old-datetime-tz) + days,
                               MTIME (old-datetime-tz) + milliseconds,
                               TIMEZONE(old-dateime-tz) ).
```

  The DATETIME-TZ function ensures that the time portion remains within the valid range, by adding day(s) to the date portion when the time part goes over the number of milliseconds in a day.

See also

+ Date addition operator, - Datetime subtraction operator, ADD-INTERVAL function, DATETIME function, DATETIME-TZ function

+= Addition assignment operator

Adds a value to an ABL data element. Provides a shorter syntax for applying concatenation, numeric addition, and date-based addition.

Syntax

```
field += expression [ NO-ERROR ]
```

`field += expression` is equivalent to `field = field + (expression)`.

Note: Spaces are required around the assignment operator. You cannot write `field+=expression`.

Note: This syntax can also be used in the `ASSIGN`, `BUFFER-COPY`, `SET`, and `UPDATE` statements.

field

The name of an ABL data element to which you want to add the value of *expression*, and that is defined with a data type that is compatible with the data type of *expression*. Valid data types are INTEGER, INT64, DECIMAL, CHARACTER, LONGCHAR, DATE, DATETIME, and DATETIME-TZ. The data element can include:

- A database or temp-table field
- A variable scoped to the current procedure, user-defined function, or method of a class, or an accessible class-based variable data member, including a subscripted array variable
- A parameter defined for the current procedure, user-defined function, or method of a class, including a subscripted array parameter
- A writable class-based or COM property, including a subscripted array property
- A writable handle attribute or system handle attribute
- ABL syntax that specifies the CURRENT-VALUE statement, DYNAMIC-CURRENT-VALUE statement, DYNAMIC-PROPERTY statement, ENTRY statement, EXTENT statement, LENGTH statement, PROPATH statement, or SUBSTRING statement

expression

An expression with a data type that is consistent with the data type of *field*.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages. With the NO-ERROR option, if ERROR is raised, then the ABL element on the left-hand side of the assignment is unchanged.

Note: For more information on the rules related to assignment, see the notes section in the Assignment (=) statement reference entry.

Examples

The following example shows how to do string concatenation using the += assignment operator. Note that the data types (CHAR and LONGCHAR) are compatible.

```
VAR LONGCHAR str1 = "abc".
VAR CHAR str2 = "def".

str1 += str2.  // abcdef

/* Equivalent statement
str1 = str1 + str2.  */
```

The following example shows how to do integer addition using the += assignment operator.

```
VAR INT i1 = 100.
VAR INT i2 = 50.

i1 += i2.  // 150

/* Equivalent statement
i1 = i1 + i2.        */
```

The following example uses the += assignment operator on variables with mixed numeric data types.

```
VAR INT i1 = 10.
VAR INT64 i2 = 1000.
VAR DECIMAL d1 = 20.5.

i1 += i2 + d1.  // 1031

/* Equivalent statement
i1 = i1 + (i2 + d1). */
```

The following example uses the += assignment operator to add 7 days to a DATE variable.

```
VAR DATE date1 = 06/01/2020.

date1 += 7.  // 06/08/2020

/* Equivalent statement
date1 = date1 + 7.   */
```

The following example uses the += assignment operator to update a database field.

```
VAR INT extraDiscount = 300.

FIND FIRST customer.
customer.discount += extraDiscount.

/* Equivalent statement
customer.discount = customer.discount + extraDiscount. */
```

The following example uses the += assignment operator to update a class property:

```
VAR ClassA myObj.
VAR INT myValue = 10.

myObj = NEW ClassA().
myObj:MyIntProp  = 250.
myObj:MyIntProp += myValue.  // 260

/* Equivalent statement
myObj:MyIntProp = myObj:MyIntProp + myValue. */
```

The following example code uses the += assignment operator, within an `ASSIGN` statement, to update the `CreditLimit` field and the `myCtr` variable.

```
VAR INT myCtr = 0.

FOR EACH Customer WHERE NAME BEGINS "K":
    ASSIGN
        CreditLimit += 1000
        myCtr += 1.
END.

/* Equivalent statement
ASSIGN CreditLimit = CreditLimit + 1000 myCtr = myCtr + 1. */
```

The following example code uses the += assignment operator to add to the PROPATH.

```
PROPATH += ",/myNewAppDir1,/myNewAppDir2".
```

See also

Assignment (=) statement, ASSIGN statement, + Addition operator, + Concatenation operator, + Date addition operator, + Datetime addition operator

- Unary negative operator

Reverses the sign of a numeric expression. Do not confuse this operator with the subtraction operator that subtracts one expression from another.

Syntax

```
- expression
```

expression

An expression whose value is numeric.

Example

If you supply a negative value for the variable fx, the r-uneg.p procedure uses the unary negative operator (-) to reverse the sign of fx, producing the absolute value of fx (fabs-x).

r-uneg.p

```
DEFINE VARIABLE fx     AS DECIMAL NO-UNDO LABEL "X".
DEFINE VARIABLE fabs-x AS DECIMAL NO-UNDO LABEL "ABS(X)".

REPEAT:
  SET fx.
  IF fx < 0 THEN
   fabs-x = -fx.
  ELSE
    fabs-x = fx.
  DISPLAY fabs-x.
END.
```

See also

+ Unary positive operator

- Subtraction operator

Subtracts one numeric expression from another numeric expression.

Syntax

```
expression - expression
```

expression

An expression with a numeric value.

Example

The `r-subt.p` procedure determines the amount of inventory available by subtracting the amount allocated from the total on hand.

r-subt.p

```
DEFINE VARIABLE free-stock NO-UNDO LIKE on-hand LABEL "Free Stock".

FOR EACH Item NO-LOCK:
  free-stock = Item.OnHand - Item.Allocated.
  DISPLAY Item.ItemNum Item.ItemName Item.OnHand Item.Allocated free-stock.
END.
```

Note

Subtracting one decimal expression from another produces a DECIMAL value. Subtracting one INTEGER expression from another produces an INTEGER. Subtracting an integer expression (INTEGER or INT64) from a decimal expression (or subtracting a decimal expression from an integer expression) produces a DECIMAL value. Subtracting a mix of INTEGER and INT64 expressions produces an INT64 value.

See also

+ Addition operator

- Date subtraction operator

Subtracts a number of days from a date to produce a date result, or subtracts one date from another to produce an INTEGER result that represents the number of days between the two dates.

Syntax

```
date - { days | date }
```

date

An expression that evaluates to a DATE value.

days

An expression with a value of the number of days you want to subtract from *date*.

Example

This procedure finds all unshipped orders. If the promised date is more than one week ago, the procedure finds the customers who placed the order and displays the order and customer data.

r-dsub.p

```
DISPLAY "ORDERS SCHEDULED TO SHIP MORE THAN ONE WEEK LATE".
FOR EACH Order NO-LOCK WHERE Order.ShipDate = ?:
  IF (TODAY - 7) > Order.PromiseDate THEN
    DISPLAY Order.OrderNum Order.CustNum Order.PromiseDate
      (TODAY - Order.PromiseDate) LABEL "Days Late".
END.
```

See also

+ Date addition operator, - Datetime subtraction operator, ADD-INTERVAL function, DATE function, INTERVAL function

---

- Datetime subtraction operator

Subtracts a number of milliseconds from a DATETIME or a DATETIME-TZ to produce another DATETIME or DATETIME-TZ, or subtracts one DATETIME or DATETIME-TZ from another to produce an INT64 result in milliseconds.

Syntax

```
datetime - { milliseconds | datetime }
```

```
datetime-tz - { milliseconds | datetime-tz }
```

datetime

An expression that evaluates to a DATETIME value.

milliseconds

An expression that evaluates to an integer value specifying a number of milliseconds.

datetime-tz

An expression that evaluates to a DATETIME-TZ value.

Example

This example returns the date and time exactly 24 hours ago (`dtTime`). It then recalculates the number of hours (`iHours`) from the number of milliseconds (`iMsec`) since then and displays the result:

```
DEFINE VARIABLE dtTime AS DATETIME NO-UNDO.
DEFINE VARIABLE iMsec  AS INT64    NO-UNDO.
DEFINE VARIABLE iHour  AS INTEGER  NO-UNDO INITIAL 3600000.
DEFINE VARIABLE fHours AS DECIMAL  NO-UNDO.

ASSIGN
  dtTime = NOW - (24 * iHour)
  iMsec  = NOW - DATETIME-TZ(dtTime)
  fHours = iMsec / iHour.

MESSAGE "A day earlier: " dtTime "  "
        "Current hours since then: " fHours VIEW-AS ALERT-BOX.
```

Notes

- To get the number of days between two DATETIME or DATETIME-TZ variable values, use the DATE function. For example:

  ```
  num-days = DATE(dt2) - DATE(dt1)
  ```

  This operation does not take the time portion into account.

  To ensure the correct result when working with two DATETIME-TZ values, convert one of the values to the time zone of the other. For example:

  ```
  ASSIGN
    temp-dttz           = dt1
    TIMEZONE(temp-dttz) = TIMEZONE(dt2)
    num-days            = DATE(dt2) - DATE(temp-dttz).
  ```

- To subtract a specific number of days and milliseconds from a DATETIME, use the DATETIME function. For example:

  ```
  new-datetime = DATETIME( DATE(old-datetime) - days,
                           MTIME (old-datetime) - milliseconds ).
  ```

  The DATETIME function ensures the time portion remains within a valid range by borrowing a day from the date portion, when necessary.

- To subtract a specific number of days and milliseconds from a DATETIME-TZ, use the DATETIME-TZ function. For example:

  ```
  new-datetime-tz = DATETIME-TZ( DATE(old-datetime-tz) - days,
                                 MTIME (old-datetime-tz) - milliseconds,
                                 TIMEZONE(old-dateime-tz) ).
  ```

The DATETIME-TZ function ensures the time portion remains within a valid range by borrowing a day from the date portion, when necessary.

See also

- Date subtraction operator, + Datetime addition operator, ADD-INTERVAL function, DATETIME function, DATETIME-TZ function, INTERVAL function

-= Subtraction assignment operator

Subtracts a value from an ABL data element. Provides a shorter syntax for applying numeric and date-based subtraction.

Syntax

```
field -= expression [ NO-ERROR ]
```

`field -= expression` is equivalent to `field = field - (expression)`.

Note: Spaces are required around the assignment operator. You cannot write `field-=expression`.

Note: This syntax can also be used in the `ASSIGN`, `BUFFER-COPY`, `SET`, and `UPDATE` statements.

field

The name of an ABL data element from which you want to subtract the value of *expression*, and that is defined with a data type that is compatible with the data type of *expression*. Valid data types are INTEGER, INT64, DECIMAL, DATE, DATETIME, and DATETIME-TZ. The data element can include:

- A database or temp-table field
- A variable scoped to the current procedure, user-defined function, or method of a class, or an accessible class-based variable data member, including a subscripted array variable
- A parameter defined for the current procedure, user-defined function, or method of a class, including a subscripted array parameter
- A writable class-based or COM property, including a subscripted array property
- A writable handle attribute or system handle attribute
- ABL syntax that specifies the `CURRENT-VALUE` statement, `DYNAMIC-CURRENT-VALUE` statement, `DYNAMIC-PROPERTY` statement, `EXTENT` statement, or `LENGTH` statement

expression

An expression with a data type that is consistent with the data type of *field*.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages. With the NO-ERROR option, if ERROR is raised, then the ABL element on the left-hand side of the assignment is unchanged.

Note: For more information on the rules related to assignment, see the notes section in the Assignment (=) statement reference entry.

Examples

The following example shows how to do integer subtraction using the -= assignment operator.

```
VAR INT i1 = 100.
VAR INT i2 = 25.

i1 -= i2.  // 75

/* Equivalent statement
i1 = i1 - i2.        */
```

The following example uses the -= assignment operator on variables with mixed numeric data types.

```
VAR INT64 i1 = 1000.
VAR INT i2 = 100.
VAR DECIMAL d1 = 20.5.

i1 -= i2 + d1.  //  880

/* Equivalent statement
i1 = i1 - (i2 + d1). */
```

The following example uses the -= assignment operator to subtract 7 days from a DATE variable.

```
VAR DATE date1 = 07/14/2020.

date1 -= 7. // 07/07/2020

/* Equivalent statement
date1 = date1 - 7.   */
```

The following example uses the -= assignment operator to update a database field.

```
VAR DECIMAL limitDecrease = 5000.

FIND FIRST Customer.
Customer.CreditLimit -= limitDecrease.

/* Equivalent statement
Customer.CreditLimit = Customer.CreditLimit - limitDecrease. */
```

The following example uses the -= assignment operator to update a class property:

```
VAR ClassA myObj.
VAR INT myValue = 10.

myObj = NEW ClassA().
myObj:MyIntProp  = 250.
myObj:MyIntProp -= myValue.  // 240

/* Equivalent statement
myObj:MyIntProp = myObj:MyIntProp - myValue. */
```

See also

Assignment (=) statement, ASSIGN statement, - Subtraction operator, - Date subtraction operator, - Datetime subtraction operator

---

* Multiplication operator

Multiplies two numeric expressions.

Syntax

```
expression * expression
```

expression

An expression with a numeric value.

Example

This procedure computes the value of the on-hand inventory for each item. If the on-hand inventory is negative, the procedure sets the inventory value to 0.

r-mult.p

```
DEFINE VARIABLE inv-value AS DECIMAL NO-UNDO LABEL "VALUE".

FOR EACH Item NO-LOCK:
  inv-value = Item.OnHand * Item.Price.
  IF inv-value < 0 THEN
    inv-value = 0.
  DISPLAY Item.ItemNum Item.ItemName Item.OnHand Item.Price inv-value.
END.
```

Note

Multiplying two decimal expressions produces a DECIMAL value. Multiplying two INTEGER expressions produces an INTEGER value. Multiplying two INT64 expressions produces an INT64 value. Multiplying an integer expression (INTEGER or INT64) and a decimal expression produces a DECIMAL value. Multiplying an INTEGER expression and an INT64 expression produces an INT64 value.

See also

/ Division operator

*= Multiplication assignment operator

Multiplies an ABL data element by a numeric expression. Provides a shorter syntax for applying multiplication.

Syntax

```
field *= expression [ NO-ERROR ]
```

`field *= expression` is equivalent to `field = field * (expression)`.

Note: Spaces are required around the assignment operator. You cannot write `field*=expression`.

Note: This syntax can also be used in the `ASSIGN`, `BUFFER-COPY`, `SET`, and `UPDATE` statements.

field

The name of an ABL data element by which you want to multiply the value of *expression*, and that is defined with a data type that is compatible with the data type of *expression*. Valid data types are INTEGER, INT64, and DECIMAL. The data element can include:

- A database or temp-table field
- A variable scoped to the current procedure, user-defined function, or method of a class, or an accessible class-based variable data member, including a subscripted array variable
- A parameter defined for the current procedure, user-defined function, or method of a class, including a subscripted array parameter
- A writable class-based or COM property, including a subscripted array property
- A writable handle attribute or system handle attribute
- ABL syntax that specifies the `CURRENT-VALUE` statement, `DYNAMIC-CURRENT-VALUE` statement, `DYNAMIC-PROPERTY` statement, `EXTENT` statement, or `LENGTH` statement

expression

An expression with a data type that is consistent with the data type of *field*.

`NO-ERROR`

The NO-ERROR option is used to prevent the statement from raising `ERROR` and displaying error messages. With the `NO-ERROR` option, if `ERROR` is raised, then the ABL element on the left-hand side of the assignment is unchanged.

Note: For more information on the rules related to assignment, see the notes section in the Assignment (=) statement reference entry.

Examples

The following example shows how to do integer multiplication using the *= assignment operator.

```
VAR INT i1 = 100.
VAR INT i2 = 50.

i1 *= i2.  // 5000

/* Equivalent statement
i1 = i1 * i2.        */
```

The following example uses the *= assignment operator on variables with mixed numeric data types.

```
VAR INT i1 = 10.
VAR INT64 i2 = 1000.
VAR DECIMAL d1 = 20.55.

i1 *= i2 + d1.  // 10206

/* Equivalent statement
i1 = i1 * (i2 + d1). */
```

The following example uses the *= assignment operator to update a database field.

```
VAR DECIMAL limitIncrease = 1.10.

FIND FIRST customer.
Customer.CreditLimit *= limitIncrease.

/* Equivalent statement
Customer.CreditLimit = Customer.CreditLimit * limitIncrease. */
```

The following example uses the *= assignment operator to update a class property:

```
VAR ClassA myObj.
VAR INT myValue = 10.

myObj = NEW ClassA().
myObj:MyIntProp  = 250.
myObj:MyIntProp *= myValue.  // 2500

/* Equivalent statement
myObj:MyIntProp = myObj:MyIntProp * myValue. */
```

See also

Assignment (=) statement, ASSIGN statement, * Multiplication operator

/ Division operator

Divides one numeric expression by another numeric expression, producing a decimal result. This is the case for both INTEGER and INT64 expressions.

Syntax

```
expression / expression
```

expression

An expression that evaluates to a numeric value.

Example

This procedure divides the number of items allocated by the number of items on hand, producing a decimal value. The multiplication operator (*) converts that decimal value to a percentage.

r-div.p

```
DISPLAY "INVENTORY COMMITMENTS AS A PERCENT OF UNITS ON HAND".
FOR EACH Item NO-LOCK:
  DISPLAY Item.ItemNum Item.ItemName Item.Allocated Item.OnHand
    (Item.Allocated / Item.OnHand) * 100 FORMAT ">>9" LABEL "PCT".
END.
```

Notes

- ABL **always** performs division as a decimal operation (the product of 5 / 2 is 2.5, not 2). If you assign the result to an integer field, ABL rounds the decimal to make the assignment. When you want ABL to truncate a quotient to an integer, use the TRUNCATE function (TRUNCATE(5 / 2, 0) is 2).
- The result of dividing a number by 0 is the Unknown value (?), and the AVM does not display an error message.

See also

* Multiplication operator

/= Division assignment operator

Divides an ABL data element by a numeric expression. Provides a shorter syntax for applying division.

Syntax

```
field /= expression [ NO-ERROR ]
```

`field /= expression` is equivalent to `field = field / (expression)`.

Note: Spaces are required around the assignment operator. You cannot write `field/=expression`.

Note: This syntax can also be used in the `ASSIGN`, `BUFFER-COPY`, `SET`, and `UPDATE` statements.

field

The name of an ABL data element by which you want to divide the value of *expression*, and that is defined with a data type that is compatible with the data type of *expression*. Valid data types are INTEGER, INT64, and DECIMAL. The data element can include:

- A database or temp-table field
- A variable scoped to the current procedure, user-defined function, or method of a class, or an accessible class-based variable data member, including a subscripted array variable
- A parameter defined for the current procedure, user-defined function, or method of a class, including a subscripted array parameter
- A writable class-based or COM property, including a subscripted array property
- A writable handle attribute or system handle attribute
- ABL syntax that specifies the CURRENT-VALUE statement, DYNAMIC-CURRENT-VALUE statement, DYNAMIC-PROPERTY statement, EXTENT statement, or LENGTH statement

expression

An expression with a data type that is consistent with the data type of *field*.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages. With the NO-ERROR option, if ERROR is raised, then the ABL element on the left-hand side of the assignment is unchanged.

Note: For more information on the rules related to assignment, see the notes section in the Assignment (=) statement reference entry.

Examples

The following example shows how to do integer division using the /= assignment operator.

```
VAR INT i1 = 100.
VAR INT i2 = 50.

i1 /= i2.  // 2

/* Equivalent statement
i1 = i1 / i2.        */
```

The following example uses the /= assignment operator on variables with mixed numeric data types.

```
VAR INT64 i1 = 1000.
VAR INT i2 = 100.
VAR DECIMAL d1 = 50.5.

i1 /= i2 + d1.  // 7

/* Equivalent statement
i1 = i1 / (i2 + d1). */
```

The following example uses the /= assignment operator to update a database field.

```
VAR DECIMAL limitDecrease = 2.

FIND FIRST customer.
Customer.CreditLimit /= limitDecrease.

/* Equivalent statement
Customer.CreditLimit = Customer.CreditLimit / limitDecrease. */
```

The following example uses the /= assignment operator to update a class property:

```
VAR ClassA myObj.
VAR INTEGER myValue = 10.

myObj = NEW ClassA().
myObj:MyIntProp  = 250.
myObj:MyIntProp /= myValue.  // 25

/* Equivalent statement
myObj:MyIntProp = myObj:MyIntProp / myValue. */
```

See also

Assignment (=) statement, ASSIGN statement, / Division operator

ABSOLUTE function

Returns the absolute value of a numeric value.

Syntax

```
ABSOLUTE ( n )
```

n

An integer or decimal expression. The return value is the same format as *n*.

Example

This procedure calculates the number of miles you drive between highway exit ramps.

r-abs.p

```
DEFINE VARIABLE mark-start  AS DECIMAL NO-UNDO.
DEFINE VARIABLE mark-finish AS DECIMAL NO-UNDO.
DEFINE VARIABLE units       AS LOGICAL NO-UNDO FORMAT "miles/kilometers".

FORM
  mark-start LABEL "Mile marker for highway on-ramp" SKIP
  mark-finish LABEL "Mile marker next to your exit" SKIP(1)
  units LABEL "Measure in <m>iles or <k>ilometers" SKIP(1)
  WITH FRAME question SIDE-LABELS
  TITLE "This program calculates distance driven.".

UPDATE mark-start mark-finish units WITH FRAME question.

DISPLAY
  "You have driven" ABSOLUTE(mark-start - mark-finish) units
  WITH NO-LABELS FRAME answer.
```

ACCUM function

Returns the value of an aggregate expression that is calculated by an ACCUMULATE or aggregate phrase of a DISPLAY statement.

Syntax

```
ACCUM aggregate-phraseexpression
```

aggregate-phrase

A phrase that identifies the aggregate value it should return. This is the syntax for *aggregate-phrase*:

```
{ AVERAGE | COUNT | MAXIMUM | MINIMUM | TOTAL | SUB-AVERAGE
  | SUB-COUNT | SUB-MAXIMUM | SUB-MINIMUM | SUB-TOTAL }
  [ BY break-group]
```

For more information on aggregate items, see the Aggregate phrase reference entry.

expression

An expression that was used in an earlier ACCUMULATE or DISPLAY statement. The expression you use in the ACCUMULATE or DISPLAY statement and the expression you use in the ACCUM function must be in exactly the same form. (For example, "on-hand * cost" and "cost * on-hand" are not in exactly the same form.) For the AVERAGE, SUB-AVERAGE, TOTAL, and SUB-TOTAL aggregate phrases, *expression* must be numeric.

Example

This procedure shows a total for the extended price of each item on an order. The running total of the order is displayed as well as the order total and grand total for all orders. This procedure accumulates totals at three levels.

r-accum.p

```
FOR EACH Order NO-LOCK:
  DISPLAY Order.OrderNum Order.CustNum Order.OrderDate Order.PromiseDate
    Order.ShipDate.
  FOR EACH OrderLine OF Order NO-UNDO:
    DISPLAY OrderLine.LineNum OrderLine.ItemNum OrderLine.Qty
      OrderLine.Price (OrderLine.Qty * OrderLine.Price) LABEL "Ext Price".
    ACCUMULATE OrderLine.Qty * OrderLine.Price (TOTAL).
    DISPLAY (ACCUM TOTAL OrderLine.Qty * OrderLine.Price) LABEL "Accum Total".
  END.
  DISPLAY (ACCUM TOTAL OrderLine.Qty * OrderLine.Price) LABEL "Total".
END.
DISPLAY (ACCUM TOTAL OrderLine.Qty * OrderLine.Price) LABEL "Grand Total"
  WITH ROW 1.
```

See also

ACCUMULATE statement, AGGREGATE statement, DISPLAY statement

ACCUMULATE statement

Calculates one or more aggregate values of an expression during the iterations of a block. Use the ACCUM function to access the result of this accumulation.

Syntax

```
ACCUMULATE { expression ( aggregate-phrase ) } ...
```

expression

An expression for which you want to calculate the aggregate value. The expression you use in the ACCUMULATE statement and the expression you use in the ACCUM function (when using the result of the ACCUMULATE statement) must be in exactly the same form. (For example, "A * B" and "B * A" are not in exactly the same form.)

aggregate-phrase

Identifies one or more values to calculate based on a change in expression or a break group. This is the syntax for *aggregate-phrase*:

Syntax

```
{ AVERAGE | COUNT | MAXIMUM | MINIMUM | TOTAL | SUB-AVERAGE
  | SUB-COUNT | SUB-MAXIMUM | SUB-MINIMUM | SUB-TOTAL }...
  [ BY break-group ]...
```

For more information, see the Aggregate phrase reference entry.

Examples

This procedure calculates and displays statistics for all customers, but does not show the detail for each customer.

r-acmlt.p

```
FOR EACH Customer NO-LOCK:
  ACCUMULATE Customer.CreditLimit (AVERAGE COUNT MAXIMUM).END.

DISPLAY "MAX-CREDIT STATISTICS FOR ALL CUSTOMERS:" SKIP(2)
        "AVERAGE =" (ACCUM AVERAGE Customer.CreditLimit) SKIP(1)
        "MAXIMUM =" (ACCUM MAXIMUM Customer.CreditLimit) SKIP(1)
        "NUMBER OF CUSTOMERS =" (ACCUM COUNT Customer.CreditLimit) SKIP(1)
        WITH NO-LABELS.
```

The following procedure lists each item with its inventory value and lists that value as a percentage of the total inventory value of all items; it sorts items by highest value.

r-acmlt2.p

```
FOR EACH Item NO-LOCK:
  ACCUMULATE Item.OnHand * Item.Price (TOTAL).
END.

FOR EACH Item NO-LOCK BY Item.OnHand * Item.Price DESCENDING:
  DISPLAY Item.ItemNum Item.OnHand Item.Price Item.OnHand * Item.Price
    LABEL "Value" 100 * (Item.OnHand * Item.Price) /
    (ACCUM TOTAL Item.OnHand * Item.Price) LABEL "Value %".
END.
```

The following procedure displays all customers, sorted by salesrep and country within the list for each salesrep. The procedure calculates the balance for each customer, total balance for each country, and total balance for each salesrep.

r-acc.p

```
FOR EACH Customer NO-LOCK BREAK BY Customer.SalesRep BY Customer.Country:
  ACCUMULATE Customer.Balance
    (TOTAL BY Customer.SalesRep BY Customer.Country).
  DISPLAY Customer.SalesRep WHEN FIRST-OF(Customer.SalesRrep)
    Customer.Country Customer.Name Customer.Balance.
  IF LAST-OF(Customer.Country) THEN
    DISPLAY ACCUM TOTAL BY Customer.Country Customer.Balance
      COLUMN-LABEL "Country!Total".
  IF LAST-OF(Customer.SalesRep) THEN DO:
    DISPLAY Customer.SalesRep ACCUM TOTAL BY Customer.SalesRep
      Customer.Balance COLUMN-LABEL "SalesRep!Total".
    DOWN 1.
  END.
END.
```

Note

You can use the ACCUMULATE statement only in blocks with the implicit looping property. ABL automatically supplies looping services to REPEAT and FOR EACH blocks. See *Develop ABL Applications* for more information on block properties.

See also

ACCUM function, Aggregate phrase, AGGREGATE statement

ACTIVE-FORM system reference
(Windows only; GUI for .NET only)

An object reference to the last .NET form or ABL window to receive focus. The object reference returned by ACTIVE-FORM has the OpenEdge .NET Progress.Windows.IForm interface type.

Syntax

```
ACTIVE-FORM
```

Notes

- ACTIVE-FORM references instances of any class that ABL recognizes as a form, including the following form classes:
  - Progress.Windows.Form class — A custom OpenEdge .NET form class that encapsulates a Microsoft .NET `System.Windows.Forms.Form` for use within an ABL session. If a .NET form is last to receive focus and you use the `Progress.Windows.Form` class to instantiate all .NET forms, ACTIVE-FORM references the affected `Progress.Windows.Form` instance.
  - Progress.Windows.FormProxy class — An OpenEdge .NET form class that is associated with each ABL window instantiated in any ABL session that references .NET objects. If an ABL window is last to receive focus, ACTIVE-FORM references the `Progress.Windows.FormProxy` object associated with the affected window.
- If the last ABL window to receive focus is the default window, ACTIVE-FORM returns the Unknown value (?), because ABL does not create a `FormProxy` object for the default window.

- You cannot use ACTIVE-FORM if the window with focus is not a `Progress.Windows.Form` or a native ABL window. If you directly instantiate .NET forms using `System.Windows.Forms.Form` (or any other native .NET form class), you must use the .NET `ActiveForm` property to identify the native .NET form instance that has focus. `Progress.Windows.Form` also inherits the `ActiveForm` property from `System.Windows.Forms.Form`, and this property also references the .NET native form instance that a given `Progress.Windows.Form` object represents. However, `Progress.Windows.FormProxy` **does not** inherit the `ActiveForm` property and does not provide a meaningful reference when an ABL window has focus.

See also

ACTIVE-WINDOW system handle, Progress.Windows.Form class, Progress.Windows.FormProxy class

---

ADD-INTERVAL function

Adds a time interval to, or subtracts a time interval from, a DATE, DATETIME, or DATETIME-TZ value, and returns the new value.

Syntax

```
ADD-INTERVAL (datetime, interval-amount, interval-unit)
```

datetime

An expression whose value is a DATE, DATETIME, or DATETIME-TZ.

interval-amount

A signed integer (positive or negative) indicating the amount of time you want to add to or subtract from *datetime* value.

interval-unit

A character constant, or a character expression that evaluates to one of the following time units: ‘years', ‘months', ‘weeks', ‘days', 'hours', ‘minutes', ‘seconds' or ‘milliseconds'. These values are case insensitive and may be singular.

Note

To add or subtract months or years, this function converts the date to Gregorian before adding or subtracting the year or month value. If the result is an invalid date, the function decrements the day part of the date until a valid date is obtained. For example:

- Adding 1 month to January 30, 2003 yields February 28, 2003
- Adding 13 months to January 30, 2003 yields February 29, 2004 (2004 is a leap year)
- Subtracting 1 month from December 31, 2003 yields November 30, 2003

See also

+ Date addition operator, - Date subtraction operator, + Datetime addition operator, - Datetime subtraction operator, DATE function, DATETIME function, DATETIME-TZ function, INTERVAL function

Aggregate phrase

Identifies one or more values to calculate based on a change in an expression or a break group.

Syntax

```
{   AVERAGE
  | COUNT
  | MAXIMUM
  | MINIMUM
  | TOTAL
  | SUB-AVERAGE
  | SUB-COUNT
  | SUB-MAXIMUM
  | SUB-MINIMUM
  | SUB-TOTAL
}... [ LABEL aggr-label ] [ BY break-group ]...
```

AVERAGE

Calculates the average of all of the values of the expression in a break group and the average of all of the values of the expression in all break groups.

COUNT

Calculates the number of times the expression was counted in a break group and the count of all the values in all break groups.

MAXIMUM

Calculates the maximum of all of the values of the expression in a break group and the maximum of all the values of the expression in all break groups.

MINIMUM

Calculates the minimum of all of the values of the expression in a break group and the minimum of all the values of the expression in all break groups.

TOTAL

Calculates the subtotal of all of the values of the expression in a break group and the grand total of all of the values of the expression in all break groups. When you use default aggregates, the actual display of the grand total is deferred until the frame goes out of scope.

SUB-AVERAGE

Averages values in a break group. Does not supply an average for all records, just for those in each break group.

SUB-COUNT

Counts the number of times an expression is in a break group. Does not supply a count for all records, just for those in each break group.

SUB-MAXIMUM

Shows the maximum value of an expression in a break group. Does not supply a maximum value for all records, just for those in each break group.

SUB-MINIMUM

Shows the minimum value of an expression in a break group. Does not supply a minimum value for all records, just for those in each break group.

SUB-TOTAL

Subtotals all of the values of the expression in a break group. Does not supply a total value for all records, just for those in each break group.

BY *break-group*

Performs aggregation for break groups if you use the BREAK option in a FOR EACH block header.

LABEL *aggr-label*

Specifies a label for the aggregate value. *aggr-label* is a standard ABL string and can use a string attribute. You can specify a maximum length attribute that is greater than the length of the longest label translation.

Examples

This procedure lists the customer information for all customers (categorized by country) and a subtotal of each country's balance. If you use TOTAL instead of SUB-TOTAL, the AVM displays a grand total.

r-aggreg.p

```
FOR EACH Customer NO-LOCK BREAK BY Customer.Country:
  DISPLAY Customer.Name Customer.Country Customer.Balance
    (SUB-TOTAL BY Customer.country).
END.
```

In the following procedure, the AVM displays the result of the COUNT aggregate even though no accumulation has occurred. In this example, COUNT displays as 0.

r-agcnt.p

```
DEFINE VARIABLE prntr AS LOGICAL NO-UNDO.

FOR EACH Item NO-LOCK:
  IF prntr THEN
    DISPLAY Item.ItemName Item.Price(COUNT) WITH FRAME pr.
END.
```

In the following procedure, the AVM uses "Avg. Credit Limit" and "Max. Credit Limit" as the labels for the AVERAGE and MAXIMUM aggregates respectively.

r-aglim.p

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.Name Customer.CreditLimit
    (AVERAGE LABEL "Avg. Credit Limit"
     MAXIMUM LABEL "Max. Credit Limit"
     TOTAL) WITH FRAME frame1 12 DOWN.
 END.
```

Notes

- By default, the AVM displays the aggregate result when the aggregate group ends, as long as the block iterates. If you want to suppress automatic display of zero aggregates, use the ACCUMULATE statement to perform the calculation and test the result with the ACCUM function before displaying the result.
- When you use aggregate phrases to accumulate values within shared frames, you must include the ACCUM option in the Frame phrase. See the Frame phrase reference entry for more information.
- An Aggregate phrase is designed to generate aggregate values for blocks that read forward through records in a sequential fashion. In blocks that read records in a non-sequential fashion (for example, FIND PREV, FIND FIRST, FIND LAST, etc.), an aggregate could yield unexpected values.
- Avoid specifying more than one aggregate of the same type for a single field in a block. If an aggregate of the same type for a single field executes more than once during a single iteration of a block, the aggregate could yield unexpected value.
- The BY phrase supports aggregates on break groups. The aggregate for a break group should reside in the block that defines the break group. Avoid positioning the aggregate in a conditional statement or sub-block in the block that defines the break group. Failure to follow these guidelines may yield unexpected values for the aggregate.

  You can build your own algorithms to generate aggregates for break groups in situations that do not adhere to these guidelines. For example, you can use variables to store aggregate values for use in expressions that generate the appropriate aggregate values for break groups across blocks in a procedure.

See also

ACCUMULATE statement, FOR statement

AGGREGATE statement

Calculates a single aggregate value for a collection of rows on the database and assigns it to a variable. Because the operation executes server-side, you may see a performance improvement compared to doing the same operation client-side using the ACCUMULATE statement and ACCUM function.

Syntax

```
AGGREGATE aggregate-target-var = { aggregate-operation } (field)
  FOR table-name [ where-clause ].
```

aggregate-target-var

The name of a field or variable to set with the aggregate value. The value returned is numeric, therefore a DECIMAL variable holds any value returned. You may use an INTEGER or INT64 variable, but the value returned is rounded.

aggregate-operation

Valid values are COUNT, TOTAL and AVERAGE.

- COUNT - Counts the number of rows of *field* that satisfy the *where-clause*, if specified. Unknown (?) values are included in the count.
- TOTAL - Sums the values of *field* in all rows that satisfy the *where-clause*, if specified. *field* must be a field of some numeric data type. Unknown (?) values are not included in the total.
- AVERAGE - Calculates the average of all values of the *field* that satisfy the *where-clause*, if specified. *field* must be a field of some numeric data type. When computing the average (total/count), unknown (?) values are included in the count, but not the total. If the query returns no records for the AVERAGE calculation, the AGGREGATE statement returns the Unknown (?) value.

field

The field name of a table you want to get the aggregate for. The field name can be qualified with the table name (for example, TableName.FieldName).

FOR *table-name* [*where-clause*]

Defines the table name of the field and an optional *where-clause* for limiting the range of records to be used for the aggregate. If no *where-clause* is specified, then all values are included.

If *field* is also qualified with a table name, then the table buffers must be the same.

Examples

This example code counts the number of records in the Customer table:

```
VAR INTEGER numCustomers.

AGGREGATE numCustomers = COUNT(CustNum) FOR Customer.

MESSAGE "Number of customers: " numCustomers
  VIEW-AS ALERT-BOX.
```

This example code counts the number of customers in Boston:

```
VAR INTEGER numCustomers.

AGGREGATE numCustomers = Count(CustNum) FOR Customer
  WHERE Country EQ 'USA' AND City EQ 'Boston'.

MESSAGE "Number of customers: " numCustomers
  VIEW-AS ALERT-BOX.
```

This example code calculate the average balance for customers in Chicago:

```
VAR DECIMAL avgBalance.

AGGREGATE avgBalance = AVERAGE(Balance) FOR Customer
  WHERE Country EQ 'USA' AND City EQ 'Chicago'.

MESSAGE "Average balance: " avgBalance
  VIEW-AS ALERT-BOX.
```

This example code sums the customer balances for customers in Los Angeles:

```
VAR DECIMAL totalBalance.

AGGREGATE totalBalance = TOTAL(Balance) FOR Customer
  WHERE Country EQ 'USA' AND City EQ 'Los Angeles'.

MESSAGE "Total balance: " totalBalance
  VIEW-AS ALERT-BOX.
```

Notes

- The AGGREGATE statement accesses all records using NO-LOCK.
- The AGGREGATE statement cannot be inside a FOR loop for the same buffer.
- If the session has a record in scope, then that record is no longer in scope after the AGGREGATE statement executes. Therefore, you should not use the same buffers in AGGREGATE statements that are used in other parts of the application.
- The AGGREGATE statement can be used with self-service clients.
- The following limitations apply to the AGGREGATE statement:
  - Can only be used with a single aggregate on a single table.
  - Is not allowed for DataServers.
  - Is not supported with the TENANT-WHERE clause for multi-tenant tables.

ALIAS function

The ALIAS function returns the alias corresponding to the integer value of expression.

Syntax

```
ALIAS ( integer-expression )
```

integer-expression

If there are, for example, three currently defined aliases, the functions ALIAS(1), ALIAS(2), and ALIAS(3) return them. If the ALIAS function cannot find a defined alias, it returns the Unknown value

(?). For example, building on the previous example of three defined aliases, the functions ALIAS(4), ALIAS(5), and ALIAS(6) return the Unknown value (?) because they cannot find a defined alias.

Example

This procedure displays the aliases and logical names of all connected databases.

r-aliasf.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

REPEAT ix = 1 TO NUM-ALIASES:
  DISPLAY ALIAS(ix) LABEL "Alias"
    LDBNAME(ALIAS(ix)) LABEL "Logical Database".
END.
```

See also

CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-ALIASES function, NUM-DBS function, PDBNAME function, SDBNAME function

AMBIGUOUS function

Returns a TRUE value if the last FIND statement for a particular record found more than one record that met the specified index criteria.

Syntax

```
AMBIGUOUS record
```

record

The name of a record or record buffer used in a previous FIND statement.

To access a record in a table defined for multiple databases, you might have to qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

Example

The following example retrieves a customer record based on a name (cname) supplied by the user. If the procedure finds a record, it displays fields from that record. If it does not find a record because more than one record matched the selection criteria (name = cname), it displays the message: "There is more than one customer with that name." If it does not find a record because no records matched the selection criteria, it displays "Cannot find customer with that name".

r-ambig.p

```
DEFINE VARIABLE cName NO-UNDO LIKE Customer.Name LABEL "Cust Name".

REPEAT:
  SET cName.
  FIND Customer NO-LOCK WHERE Customer.Name = cName NO-ERROR.
  IF AVAILABLE Customer THEN
    DISPLAY Customer.CustNum Customer.Address Customer.City Customer.State
      Customer.PostalCode.
  ELSE IF AMBIGUOUS Customer THEN
    MESSAGE "There is more than one customer with that name".
  ELSE
    MESSAGE "Cannot find customer with that name".
END.
```

Sometimes the AMBIGUOUS function returns a TRUE value when there is no ambiguity. For example, if there is exactly one customer record, the following statement finds that record. Otherwise, the following statement always returns a message of "not found" rather than "ambiguous":

```
FIND Customer WHERE Customer.Name BEGINS "".
```

Additionally, the following statement succeeds if there is only one Smith listed in the database:

```
FIND Employee WHERE Employee.LastName = "Smith"
  AND Employee.FirstName BEGINS "".
```

Note

AMBIGUOUS is useful only when there is an index. If you use the AMBIGUOUS function to test a work file record, the function returns a value of FALSE because work files do not have indexes.

See also

AVAILABLE function, FIND statement, LOCKED function, NEW function (record buffers)

AND operator (bitwise)

Performs a bitwise AND operation on two expressions of the same flag enum type and returns a new instance of the same type.

Syntax

```
flag-enum-expression AND flag-enum-expression
```

flag-enum-expression

An expression whose value is a flag enum instance. Both expressions must be the same flag enum type, and neither expression can be the Unknown value (?).

Example

You can use bitwise AND to check to see if a particular flag or flags is set. This code fragment defines a variable of type `Progress.Reflect.Flags` (see the Progress.Reflect.Flags enumeration entry for a full list of members) and uses AND to check to see if the `Public` flag is set.

```
DEFINE VARIABLE flagsEnum AS Progress.Reflect.Flags.

flagsEnum = Flags:Public OR Flags:Private.

IF (flagsEnum AND Flags:Public) EQ Flags:Public THEN...
```

The bitwise AND operation results in an enum instance in which the only flags set are the flags set in both expressions. In this case, the resulting instance has the `Public` flag set.

You can also use `Progress.Lang.FlagsEnum`'s IsFlagSet( ) method to achieve the same result:

```
IF flagsEnum:IsFlagSet(Flags:Public) THEN...
```

Notes

- This operator can be used with both ABL and .NET flag enums.

See also

ENUM statement, IsFlagSet( ) method, NOT operator (bitwise), XOR operator (bitwise), OR operator (bitwise)

AND operator (logical)

Returns a TRUE value if each logical expression is TRUE.

Syntax

```
expression AND expression
```

expression

An expression that evaluates to a logical value (TRUE or FALSE).

Example

This procedure lists all customers with credit limits between two values (supplied by the user and stored in the variables low-credit and hi-credit). The expressions Customer.CreditLimit >= low-credit and Customer.CreditLimit <= hi-credit are logical expressions because each yields a true or false value. Using the AND operator to join these logical expressions results in a logical expression that follows the WHERE keyword.

r-and.p

```
DEFINE VARIABLE low-credit LIKE credit-limit LABEL "Low Credit Limit".
DEFINE VARIABLE hi-credit  LIKE credit-limit LABEL "High Credit Limit".

REPEAT:
  SET low-credit hi-credit WITH FRAME cr-range.
  FOR EACH Customer NO-LOCK WHERE
    (Customer.CreditLimit >= low-credit) AND
    (Customer.CreditLimit <= hi-credit):
    DISPLAY Customer.CustNum Customer.Name Customer.CreditLimit.
  END.
END.
```

See also

NOT operator (logical), OR operator (logical)

APPLY statement

Applies an event to a widget or procedure.

Syntax

```
APPLY event [ TO widget-phrase ]
```

event

An expression whose value is the key code or event name that you want to apply. *Event* can be either a character-string value (event name) or an integer (key code) expression. A special value of *event* is the value of the LASTKEY function. The LASTKEY function returns the keycode for the last event read from the user (that is, from the keyboard or mouse) or the last character read from an input file. *Event* can also be “ERROR”, in which case it raises error in the current block. This can be handled by an ON ERROR block or by other standard error handling techniques. *Event* can also be a developer event (U1 to U10 or CLOSE). For more information on default system actions and events, see the Handle-based Object Events Reference.

TO *widget-phrase*

Specifies a widget or procedure to which the event is applied.

Example

This procedure shows how to use the APPLY statement to create keyboard accelerators. When you run this procedure you can invoke the trigger block attached to the order-but button by choosing the button directly or by pressing **F10** in the Name field. When you press **F10**, the AVM sends the CHOOSE event to the button. This is equivalent to choosing the button with the mouse.

r-apply.p

```
DEFINE BUTTON order-but LABEL "Order"
  TRIGGERS:
    ON CHOOSE DO:
      FIND FIRST Order OF Customer NO-ERROR.
      IF AVAILABLE Order THEN
        UPDATE Order WITH FRAME upd-dlg
          VIEW-AS DIALOG-BOX TITLE "Update Order" SIDE-LABELS.
    END.
  END TRIGGERS.

FORM order-but Customer.Name WITH FRAME x.

ON F10 OF Customer.Name DO:
  APPLY "CHOOSE" TO order-but IN FRAME x.
END.

FIND FIRST Customer NO-LOCK.
DISPLAY order-but Customer.Name WITH FRAME x.
ENABLE ALL WITH FRAME x.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- You can apply any event to any widget, including an insensitive widget. Most event-widget pairs have a default system action, but a few do. For example, the default system action for the A event on a fill-in widget is to insert the letter A into the fill-in at the current cursor location; however, there is no default system action for the A event on a button widget. Also, if you APPLY an event to a button, for example, the image of the button does not "depress" and then "pop back out." Depending on the event-widget pair, the APPLY statement may or may not perform the default system action.

  Regardless of whether there is a default system action associated with an event-widget pair, you can write a trigger for the pair. The APPLY statement executes a trigger associated with an event-widget pair. If the event-widget pair has a default system action, that action occurs before or after the trigger executes, depending on the event. For more information on default system actions and events, see the next note and the Handle-based Object Events Reference.

- When, in a graphical interface, you APPLY an event to a widget, you cannot easily invoke the widget animation code that runs when the user interacts with the widget physically. For example, if you APPLY a "choose" event to a button widget, you cannot easily make the image of the button move down and up, as occurs when the user clicks on the button. The difficulty exists because the AVM does not provide access to the widget animation code, which resides in the windowing system. When the user clicks on the button, the windowing system detects the event, invokes the button animation code, perhaps performs other tasks, and passes the event to the AVM, which invokes the trigger code associated with the event. When you APPLY a "choose" event to the button, the AVM merely invokes the trigger code associated with the event. In neither case does the AVM access, or provide access to, the button animation code.

  One widget that does not have this difficulty is the fill-in. When you APPLY a character-string event to a fill-in, the character string appears in the image of the fill-in. The AVM accomplishes this by placing a copy of the character string into a buffer that maps to the same portion of the screen as the image of the fill-in.

- The APPLY statement serves as an important communications mechanism between procedures in an application. By defining triggers for events in a procedure, you can encapsulate functionality in the procedure. The APPLY statement allows you to access that encapsulated functionality from another procedure through a simple event interface.

- The APPLY statement is double-byte enabled. A character-string value specified for the *event* argument can contain double-byte characters.

- If a procedure calls another procedure from within an EDITING phrase and the called procedure uses the APPLY statement, the effect is the same as if the APPLY statement occurred directly within the EDITING phrase.
- If you are using APPLY in an EDITING phrase and *expression* is a key that causes a GO action (**GO**, or any key in a list used with the GO-ON option), the AVM does not immediately exit the EDITING phrase but instead processes all the remaining statements in the phrase. If RETRY, NEXT, UNDO RETRY, or UNDO NEXT is executed before the end of the phrase, the AVM ignores the GO and continues processing the EDITING phrase.
- APPLY -2 is the same as APPLY ENDKEY.
- For SpeedScript, you can apply an event to a procedure only.
- You cannot use the APPLY statement to apply (publish or send) an event on behalf of a .NET object. .NET provides its own mechanisms for publishing events. In some cases, this might include a method for publishing a given event on behalf of the object. For more information, see the .NET development documentation on MSDN.

See also

Developer events, ON statement, Widget phrase

ASC function

Converts a character expression representing a single character into the corresponding ASCII (or internal code page) value, returned as an INTEGER.

Syntax

```
ASC ( expression
    [ , target-codepage [ , source-codepage ] ] )
```

expression

An expression with a value of a single character that you want to convert to an ASCII (or internal code page) integer value. If *expression* is a constant, you must enclose it in quotation marks (" "). If the value of *expression* is other than a single character, ASC returns the value -1.

The values for *expression* are case sensitive. For example, ASC("a") returns a different value than ASC("A").

target-codepage

A character-string expression that evaluates to the name of a code page. The name that you specify must be a valid code page name available in the *OpenEdge-install-dir*`/convmap.cp` file (a binary file that contains all of the tables that ABL uses for character management). If you supply a non-valid name, the ASC function returns the value -1 and returns a runtime error. Before returning an integer value, the ASC function converts *expression* from *source-codepage* to *target-codepage*. The returned integer value is relative to *target-codepage*. If you do not specify *target-codepage*, the value returned is the code page identified with the Internal Code Page (`-cpinternal`) parameter.

source-codepage

A character-string expression that evaluates to the name of a code page. The name that you specify must be a valid code page name available in the *OpenEdge-install-dir*/convmap.cp file. If you supply a non-valid name, the ASC function returns the value -1. The *source-codepage* specifies the name of the code page to which *expression* is relative. The default value of *source-codepage* is the code page identified with the Internal Code Page (-cpinternal) parameter.

Example

The following procedure counts how many customers names begin with each of the letters, A-Z. It counts all other customers separately. The procedure uses the ASC function to translate a letter into an integer that it uses as an array subscript for counting.

r-asc.p

```
DEFINE VARIABLE ix   AS INTEGER NO-UNDO.
DEFINE VARIABLE jx   AS INTEGER NO-UNDO.
DEFINE VARIABLE ltrl AS INTEGER NO-UNDO EXTENT 27.

FOR EACH Customer NO-LOCK:
  ix = ASC(SUBSTRING(Customer.Name,1,1)).
  IF ix < ASC("A") or ix > ASC("Z") THEN ix = EXTENT(ltrl).
  ELSE ix = ix - ASC("A") + 1.
  ltrl[ix] = ltrl[ix] + 1.
END.

DO jx = 1 TO EXTENT(ltrl) WITH NO-LABELS USE-TEXT:
  IF jx <= 26 THEN
    DISPLAY CHR(ASC("A") + jx - 1)@ ltr-name AS CHARACTER FORMAT "x(5)".
  ELSE
    DISPLAY "Other" @ ltr-name.
  DISPLAY ltrl[jx].
END.
```

Notes

- The ASC function returns the corresponding value in the specified character set. By default, the value of SESSION:CHARSET is iso8859-1. You can set a different internal code page by specifying the Internal Code Page (-cpinternal) parameter. For more information, see *Internationalize ABL Applications*.
- The ASC function is both double-byte and Unicode enabled. If the *expression* argument yields a double-byte character or multi-byte Unicode character in the target code page, this function can return a value greater than 255.

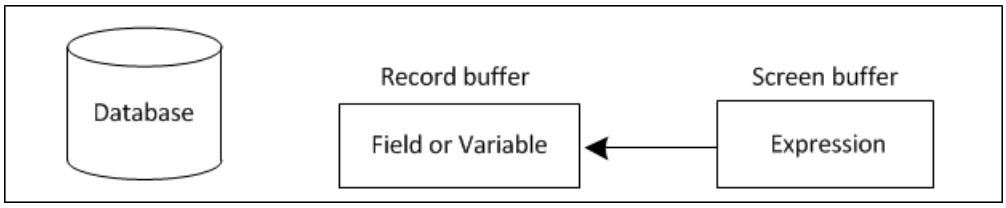
See also

CHR function, CODEPAGE-CONVERT function, INTEGER function, SESSION system handle

Assignment (=) statement

Assigns the value of an expression to a database field or variable.

Data movement



Syntax

```
field = expression [ NO-ERROR ]
```

field

The name of an ABL data element to which you want to assign the value of *expression* and that is defined with a data type that is compatible with the data type of *expression*. This data element can include a:

- Database or temp-table field
- Variable scoped to the current procedure, user-defined function, or method of a class or an accessible class-based variable data member, including a subscripted or unsubscripted array variable
- Parameter defined for the current procedure, user-defined function, or method of a class, including a subscripted or unsubscripted array parameter
- Writable class-based or COM property, including a subscripted or unsubscripted array property
- Writable handle attribute (such as PRIVATE-DATA)
- Writable system handle (such as CURRENT-WINDOW)
- ABL syntax that specifies a keyword-driven assignment statement (such as the PROPATH statement, SUBSTRING statement, or similar statement)

expression

An expression with a data type that is consistent with the data type of *field*. For more information, see the Expression reference entry.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

For the assignment (=) statement with NO-ERROR, if ERROR is raised, then the left side of the assignment will be unchanged.

Note: You can use compound assignment operators (+=, -=, *=, /=) to perform an operation and an assignment, using a shorthand notation. Supported operations include addition (numeric and date-based) and concatenation, subtraction (numeric and date-based), multiplication, and division. For more information, see += Addition assignment operator, -= Subtraction assignment operator, *= Multiplication assignment operator, and /= Division assignment operator.

Example

This procedure resets all the monthly quota values to 2500 in all salesrep records. If you want to set values for individual array elements, you can do so by making an explicit assignment using the assignment statement (=) and a specific array reference, such as month-quota[1] or month-quota[i].

r-asgmnt.p

```
DEFINE VARIABLE ctr AS INTEGER NO-UNDO.

FOR EACH SalesRep:
  DO ctr = 1 TO 12:
    SalesRep.MonthQuota[ctr] = 2500.
  END.
END.
```

Notes

- If *field expression* is a decimal, the AVM rounds the value of the expression before assigning it. If *field* is a decimal and *expression* is a decimal, the AVM rounds the value of the expression to the number of decimal places defined for the field in the Data Dictionary, or defined or implied for a variable or temp-table field.
- If *field* is an ABL array type and *expression* is not an array, and you do not identify a particular array element, the AVM stores is an integer and *expression* in each element of the array. If you identify a particular element, the AVM stores *expression* in the specified array element.
- If both *field* and *expression* are ABL array types, the AVM copies the data for all *expression* array elements into the corresponding elements of the *field* array. This is known as a *deep copy*. For more information, see Work with one-dimensional arrays in *Basic ABL*.
- ABL allows you to assign ABL arrays and .NET array objects to each other. How an array assignment works between ABL and .NET arrays depends upon the array type of *field* (the target of the assignment) and the array type of *expression* (the source for the assignment). For more information, see the Data types reference entry.
- If an assignment of one array to another encounters an error after some, but not all of the source array's elements are retrieved, none of the target's elements are updated.
- If *expression* is an ABL handle-based object (for example, a temp-table, ProDataSet, widget, or socket), *field* must be a temp-table field, variable, or other ABL data element defined as a compatible handle. In this case, the AVM assigns only the handle of the ABL handle-based object to *field*, not the entire object and its contents.
- If you assign a value to a database field, any ASSIGN trigger associated with that field executes at the end of the assignment statement (after any index changes are made). If the trigger raises ERROR, the assignment fails and the database changes are undone.
- You can embed an assignment in a SET or UPDATE statement.
- For multiple assignments, use the ASSIGN statement. This is more efficient than multiple assignment statements.
- If *field* is a handle, the *expression* on the right-hand-side must also evaluate to a handle value that is specified using an appropriate reference to a handle-based object handle. For more information on object handle references, see the Handle Attributes and Methods Reference.
- You can assign DATE, DATETIME, and DATETIME-TZ data. When the data type expression on the left side of the assignment statement contains more information than the data type expression on the right side provides (for example, *datetime-tz = date* where a DATETIME-TZ value contains more information than a DATE value), the time value defaults to midnight and the time zone is calculated using

SESSION:TIMEZONE when that attribute is in use. If the attribute is not in use, the value defaults to the session's timezone. When the data type expression on the left side of the assignment statement contains less information than the data type expression on the right side provides (for example, *date = datetime-tz* where a DATE value contains less information than a DATETIME-TZ value), the AVM converts the DATETIME-TZ value to the local date and time of the session, then drops the time and time zone.

- You can assign RAW values to MEMPTR variables and MEMPTR values to RAW variables. If the target variable is a RAW data type, the AVM re-sizes the target variable, if necessary, so that after the assignment is the same size as the source. Note that after the assignment (whether RAW = MEMPTR or MEMPTR = RAW), the target variable has a copy of the memory associated with the source—each variable has an independent copy of the data.
- You can assign large object data from one BLOB or MEMPTR to another, and from one CLOB, LONGCHAR, or CHARACTER to another. You cannot assign large object data between BLOBs and CLOBs or MEMPTRs and LONGCHARs. You can accomplish this, indirectly, by using the COPY-LOB statement. For more information, see the COPY-LOB statement reference entry.

Note: When assigning BLOB or CLOB fields, the field must appear by itself on either the right-hand or the left-hand side of the assignment.

The following table lists the default character conversions that the AVM performs when assigning CLOB, LONGCHAR, and CHARACTER data between a source and target object. References to CLOBCP and CLOBDB represent CLOB data in either the CLOB's defined code page or the database's defined code page, respectively. References to the "fixed code page" represent the code page of a target LONGCHAR variable set using the FIX-CODEPAGE statement.

Table 8: Default assignment character conversions

When the target object (on the left) is a . . .	And the source object (on the right) is a . . .	The AVM converts the data in the source object to . . .
LONGCHAR	CLOBDB	-cpinternal or the fixed code page
LONGCHAR	CLOBCP	The CLOB's defined code page or the fixed code page
LONGCHAR	CHARACTER	-cpinternal or the fixed code page
CLOBDB	CHARACTER	The database's defined code page
CLOBDB	LONGCHAR	The database's defined code page
CLOBCP	CHARACTER	The CLOB's defined code page
CLOBCP	LONGCHAR	The CLOB's defined code page
CHARACTER	CLOBDB	-cpinternal code page
CHARACTER	CLOBCP	-cpinternal code page
CHARACTER	LONGCHAR	-cpinternal code page

- When you assign the Unknown value (?) to a BLOB or CLOB field, the AVM deletes any associated object data.

- If *expression* is a solitary invocation of the NEW function (classes), this statement represents and conforms to the rules specified for the NEW statement.
- If *expression* evaluates to an object reference value, *field* must also be a data element defined as a class or interface type that is type-compatible with *expression* according to the rules for assigning references to class instances defined for the NEW statement. For more information, see the NEW statement reference entry. Thus, you can assign one object reference variable to another object reference variable when the destination object reference (on the left side of the assignment) is defined for the same class, a super class, or an interface of the object reference being assigned (on the right side of the assignment). The destination object reference retains its defined class or interface type for compilation. However, following its assignment, at run time, the destination represents the subclass of *field* (or the class that implements the interface specified by *field*) that is defined by *expression*.

  If *field* has a class type that is a subclass lower in the class hierarchy than the class type represented by *expression*, you can cast *expression* to the type of *field* using the CAST function, but only if *expression* is a super class that actually contains an instance of the *field* class type. If *field* has a class type that implements an interface type represented by *expression*, you can similarly cast *expression* using the CAST function, but only if *expression* actually contains an instance of the *field* class type. For more information about the CAST function, see the CAST function reference entry.

  After the assignment, *field* contains a copy of the object reference value returned by *expression*, which points to the same object instance, not a copy of the object referenced by *expression*.
- Although you can assign an object reference to a temp-table field defined as a Progress.Lang.Object class type, you cannot assign an object reference to a field in a database table. For more information, see *Develop Object-oriented ABL Applications*.

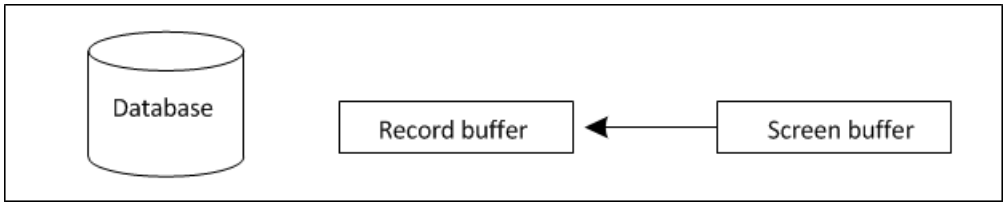
See also

ASSIGN statement, CAST function, Class-based object reference, COPY-DATASET( ) method, COPY-LOB statement, COPY-TEMP-TABLE( ) method, Data types, Expression, FIX-CODEPAGE statement, NEW statement, NO-ERROR option

ASSIGN statement

Moves data previously placed in the screen buffer by a data input statement or moves data specified within the ASSIGN statement by an expression to the corresponding fields and variables in the record buffer.

Data movement



Syntax

```
ASSIGN {
          [ [ INPUT ] FRAME frame | BROWSE browse ]
                { field [ = expression] } [ WHEN expression ]
       } ... [ NO-ERROR ]
```

```
ASSIGN { record [ EXCEPT field ... ] } [ NO-ERROR ]
```

```
[ FRAME frame | BROWSE browse ] field
```

The name of the field or variable (*field*) to be set from the corresponding value found in the screen buffer or expression. The *field* must be qualified by a frame name (*frame*) or browse name (*browse*) if *field* is specified as an input widget in more than one frame. If *field* is set from *expression*, *field* can include all of the elements that are defined for the left side of an Assignment (=) statement.

expression

An expression with a data type that is consistent with the data type of *field*. In this case, the AVM determines the *field* value from the expression rather than from the screen buffer. For more information on *expression*, see the Expression reference entry.

WHEN *expression*

Moves data to the record buffer only when *expression* has a value of TRUE. Here, *expression* is a field name, variable name, or expression whose value is logical. The AVM evaluates WHEN expressions at the beginning of the assignment, before any assignments take place.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

record

The record buffer name with the fields set, from the corresponding values in the screen buffer. Naming a record is a shorthand way to list each field in that record individually.

To use ASSIGN with a record in a table defined for multiple databases, you might have to qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

EXCEPT *field*

All fields in the record buffer are affected except for those listed. Separate field names with a space.

Note: Compound assignment operators (+=, -=, *=, /=) can be used with the ASSIGN statement. For more information, see += Addition assignment operator, -= Subtraction assignment operator, *= Multiplication assignment operator, and /= Division assignment operator.

Examples

The following procedure prompts you for a customer number and retrieves the customer record if one exists, or creates a new one if it does not exist. If it creates a new record, the value for the CustNum field is assigned from the value you entered in response to the PROMPT-FOR statement.

r-asgn.p

```
REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum NO-ERROR.
  IF NOT AVAILABLE Customer THEN DO:
    CREATE Customer.
    ASSIGN Customer.CustNum.  END.
  UPDATE Customer WITH 2 COLUMNS.
END.
```

The next procedure changes the order number and line number of an order-line record. (It copies an order-line from one order to another.) It sets the new values into variables and modifies the record with a single ASSIGN statement that contains two assignment phrases in the form *field = expression*. Thus, both fields are changed within a single statement. Because the AVM re-indexes records at the end of any statement that changes an index field value, and because OrderLine.OrderNum and OrderLine.LineNum are used jointly in one index, this technique does not generate an index until both values change.

r-asgn2.p

```
DEFINE VARIABLE neword   LIKE order-line.order-num LABEL "New Order".
DEFINE VARIABLE newordli LIKE order-line.line-num  LABEL "New Order Line".

REPEAT:
  PROMPT-FOR OrderLine.OrderNum OrderLine.LineNum.
  FIND OrderLine USING OrderLine.OrderNum AND OrderLine.LineNum.
  SET neword newordli.
  FIND Order WHERE Order.OrderNum = neword.
  ASSIGN
    OrderLine.OrderNum = neword
    OrderLine.LineNum  = newordli.
END.
```

The following example code uses the += assignment operator, with the ASSIGN statement, to update the CreditLimit field and the myCtr variable.

```
VAR INT myCtr = 0.

FOR EACH Customer WHERE NAME BEGINS "K":
    ASSIGN
        CreditLimit += 1000
        myCtr += 1.
END.

/* Equivalent statement
ASSIGN CreditLimit = CreditLimit + 1000 myCtr = myCtr + 1. */
```

Notes

- If *field* is an integer and *expression* is a decimal, the AVM rounds the value of the expression before assigning it. If *field* is a decimal and *expression* is a decimal, the AVM rounds the value of the expression

to the number of decimal places defined for the field in the Data Dictionary, or defined or implied for a variable or temp-table field.

- If *field* is an ABL array type and *expression* is not an array, and you do not identify a particular array element, the AVM stores *expression* in each element of the array. If you identify a particular element, the AVM stores *expression* in the specified array element.
- If both *field* and *expression* are ABL array types, the AVM copies the data for all *expression* array elements into the corresponding elements of the *field* array. This is known as a *deep copy*. For more information, see Work with one-dimensional arrays in *Basic ABL*.
- ABL allows you to assign ABL arrays and .NET array objects to each other. How an array assignment works between ABL and .NET arrays depends upon the array type of *field* (the target of the assignment) and the array type of *expression* (the source for the assignment). For more information, see the Data types reference entry.
- If an assignment of one array to another encounters an error after some, but not all of the source array's elements are retrieved, none of the target's elements are updated.
- If *expression* is an ABL handle-based object (for example, a temp-table, ProDataSet, widget, or socket), *field* must be a temp-table field, variable, or other ABL data element defined as a compatible handle. In this case, the AVM assigns only the handle of the ABL handle-based object to *field*, not the entire object and its contents.
- If any *field* is a field in a database record, the ASSIGN statement upgrades the record lock condition to EXCLUSIVE-LOCK before updating the record.
- If any *field* is part of a record retrieved with a field list, the ASSIGN statement rereads the complete record before updating it.
- If *field* is a handle, the *expression* on the right-hand-side of the corresponding assignment must also evaluate to a handle value that is specified using an appropriate reference to a handle-based object handle. For more information on object handle references, see the Handle Attributes and Methods Reference.
- During data entry, a validation expression defined for the field in the database or in a Format phrase executes only if the widget associated with the field receives input focus. Use the VALIDATE( ) method to execute a validation expression defined for a field regardless of whether it receives input focus or not.
- Use an ASSIGN statement after a PROMPT-FOR statement or to write changes from an enabled field to the database. ASSIGN moves the value from the screen buffer into the field or variable.
- Use the PROMPT-FOR statement to receive one or more index fields from the user, and you use the FIND statement to find a record matching those index values. If no record is found, use the CREATE statement to create a new record and use the ASSIGN statement to assign the values the user supplied to the new record.
- You cannot use the SET statement in place of the PROMPT-FOR statement. The SET statement prompts the user for input and then assigns that input to the record in the buffer. However, if there is not a record available, SET cannot assign the values.
- ASSIGN does not move data into a field or variable if there is no data in the corresponding screen field. There is data in a screen field if a DISPLAY of the field was done or if data was entered into the field. If you PROMPT-FOR a field or variable that has not been DISPLAYed in the frame and enter blanks, the AVM does not change the field or variable because it considers the screen field changed only if the data differs from what was in the field.
- If an ASSIGN statement references a field or variable that is used in more than one frame, it uses the value in the frame most recently introduced in the procedure.
- If you type blanks into a field that has never displayed data, the ENTERED function returns FALSE and the SET or ASSIGN statement does not update the underlying field or variable. Also, if the AVM marks a field as entered, and the PROMPT-FOR statement prompts for the field again and you do not enter any data, the AVM no longer considers the field entered.

- If you use a single, qualified identifier with the ASSIGN statement, the Compiler interprets the reference as *dbname.filename*. If the Compiler cannot resolve the reference as *dbname.filename*, it tries to resolve it as *filename.fieldname*.
- Many assignments within a single ASSIGN statement are more efficient than multiple ASSIGN statements. It saves r-code size and improves performance.
- The ASSIGN statement, when used in database fields, causes all related database ASSIGN triggers to execute in the order in which the fields were assigned. The ASSIGN triggers execute after all the assignments have taken place. If an ASSIGN trigger fails (or executes a RETURN statement with the ERROR option), all of the database changes are undone. See *Use ABL Database Triggers and Indexes* for more information on database triggers.
- You can assign large object data from one BLOB or MEMPTR to another, and from one CLOB, LONGCHAR, or CHARACTER to another. You cannot assign large object data between BLOBs and CLOBs or MEMPTRs and LONGCHARs. You can accomplish, indirectly, by using the COPY-LOB statement. For more information, see the COPY-LOB statement reference entry.

  The following table lists the default character conversions that the AVM performs when assigning CLOB, LONGCHAR, and CHARACTER data. References to CLOBCP and CLOBDB represent CLOB data in either the CLOB's defined code page or the database's defined code page, respectively. References to the "fixed code page" represent the code page of a target LONGCHAR variable set using the FIX-CODEPAGE statement.

  Table 9: Default character conversions with the ASSIGN statement

When the target field is a . . .	And the source expression results in a . . .	The AVM converts the result of the source expression to . . .
LONGCHAR	CLOBDB	-cpinternal or the fixed code page
LONGCHAR	CLOBCP	The CLOB's defined code page or the fixed code page
LONGCHAR	CHARACTER	-cpinternal or the fixed code page
CLOBDB	CHARACTER	The database's defined code page
CLOBDB	LONGCHAR	The database's defined code page
CLOBCP	CHARACTER	The CLOB's defined code page
CLOBCP	LONGCHAR	The CLOB's defined code page
CHARACTER	CLOBDB	-cpinternal code page
CHARACTER	CLOBCP	-cpinternal code page
CHARACTER	LONGCHAR	-cpinternal code page

- When you assign the Unknown value (?) to a BLOB or CLOB field, the AVM deletes any associated object data.
- You can assign DATE, DATETIME, and DATETIME-TZ data. When the data type expression on the left side of the assignment statement contains more information than the data type expression on the right side provides (for example, *datetime-tz = date* where a DATETIME-TZ value contains more information than a DATE value), the time value defaults to midnight and the time zone is calculated using

SESSION:TIMEZONE when that attribute is in use. If the attribute is not in use, the value defaults to the session's timezone. When the data type expression on the left side of the assignment statement contains less information than the data type expression on the right side provides (for example, *date = datetime-tz* where a DATE value contains less information than a DATETIME-TZ value), the AVM converts the DATETIME-TZ value to the local date and time of the session, then drops the time and time zone.

Note: For releases 11.6.0 and earlier, assignments such as *datetime-tz = date* used the system's timezone, regardless of whether or not SESSION:TIMEZONE was set. To revert to this earlier behavior, use the Use SESSION:TIMEZONE (-useSessionTZ) startup parameter. See *Startup Command and Parameter Reference* for more information.

- If *expression* is a solitary invocation of the NEW function, this function behaves according to the rules specified for the NEW function (classes) when not operating in the context of a NEW statement.
- If *expression* evaluates to an object reference value, *field* must also be a data element defined as a class or interface type that is type-compatible with *expression* according to the rules for assigning references to class instances defined for the NEW statement. For more information, see the NEW statement reference entry. Thus, you can assign one object reference variable to another object reference variable when the destination object reference (on the left side of the assignment) is defined for the same class, a super class, or an interface of the object reference being assigned (on the right side of the assignment). The destination object reference retains its defined class or interface type for compilation. However, following its assignment, at run time, the destination represents the subclass of *field* (or the class that implements the interface specified by *field*) that is defined by *expression*.

  If *field* has a class type that is a subclass lower in the class hierarchy than the class type represented by *expression*, you can cast *expression* to the type of *field* using the CAST function, but only if *expression* is a super class that actually contains an instance of the *field* class type. If *field* has a class type that implements an interface type represented by *expression*, you can similarly cast *expression* using the CAST function, but only if *expression* actually contains an instance of the *field* class type. For more information about the CAST function, see the CAST function reference entry.

  After the assignment, *field* contains a copy of the object reference value returned by *expression*, which points to the same object instance, not a copy of the object referenced by *expression*.
- Although you can assign an object reference to a temp-table field defined as a Progress.Lang.Object class type, you cannot assign an object reference to a field in a database table. For more information, see *Develop Object-oriented ABL Applications*.

See also

Assignment (=) statement, CAST function, Class-based object reference, COPY-DATASET( ) method, COPY-LOB statement, COPY-TEMP-TABLE( ) method, Data types, Expression, FIX-CODEPAGE statement, INPUT function, NO-ERROR option, PROMPT-FOR statement, SET statement, UPDATE statement

AT phrase

The AT phrase of the Format phrase allows explicit positioning of frame objects, either by row and column or by pixels. The AT phrase of the Frame phrase allows explicit positioning of frames with windows or parent frames.

Syntax

```
AT { COLUMN column | COLUMN-OF reference-point }
   { ROW row | ROW-OF reference-point }
   [ COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED ]
```

```
AT { X x | X-OF reference-point }
   { Y y | Y-OF reference-point }
   [ COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED ]
```

```
AT n
```

n

The column, measured in character units. This option is **not** supported for the Frame phrase. You cannot use the alignment options with this syntax. If you use this option, the AVM chooses the row based on the previous widget and form item layout of the frame. For information on form items, see the DEFINE FRAME statement or FORM statement.

COLUMN *column*

The column, measured in character units.

COLUMN-OF *reference-point*

Indicates the column position of the field relative to another field-level widget previously defined in the frame. This option is **not** supported for the Frame phrase. This is the syntax for *reference-point*:

```
widget [ { + | - } offset ]
```

In this syntax, *widget* is a reference to a field-level widget previously defined in the frame, and *offset* is a positive decimal value. For example, if *widget* is positioned at COLUMN 10, then COLUMN-OF widget + 2.5 positions the field at column 12.5.

X *x*

The X pixel coordinate.

X-OF *reference-point*

Indicates the X co-ordinate of the field relative to another field-level widget previously defined in the frame. This option is **not** supported for the Frame phrase. The co-ordinate is expressed as the co-ordinate of a widget previously defined in the frame, plus or minus an offset. The offset must be either a constant or preprocessor constant and must be a positive integer.

ROW *row*

The row, measured in character units.

ROW-OF *reference-point*

Indicates the row of the field relative to another field-level widget previously defined in the frame. This option is **not** supported for the Frame phrase. The row is expressed as the row of a widget previously defined in the frame, plus or minus an offset. The offset must be either a constant or preprocessor constant and must be a positive decimal value.

Y *y*

The Y pixel coordinate.

Y-OF *reference-point*

Indicates the Y co-ordinate of the field relative to another field-level widget previously defined in the frame. This option is **not** supported for the Frame phrase. The co-ordinate is expressed as the co-ordinate of a widget previously defined in the frame, plus or minus an offset. The offset must be either a constant or preprocessor constant and must be a positive integer.

COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED

Specifies whether to align the left edge of the field, right edge of the field, or the colon of the field label, with the specified position. This option can only be used in combination with the ROW and COLUMN options. This option is **not** supported for the Frame phrase.

Note: For r-code that is compiled with `-nocolon` in effect, the COLON-ALIGNED option will align on the last character of the field label. See the *Startup Command and Parameter Reference* for more information on `-nocolon` .

Examples

The following example uses the AT phrase to position fields within a frame.

r-at.p

```
DEFINE FRAME order-info
  Order.CustNnum AT ROW 2 COLUMN 8
  Customer.Name AT ROW 2 COLUMN 18
  Order.OrderNum AT ROW 2 COLUMN 50
  Order.OrderDate AT ROW 2 COLUMN 65
  WITH TITLE "Order Information".

 FOR EACH Order NO-LOCK BREAK BY Order.CustNum WITH FRAME order-info:
  IF FIRST-OF(Order.CustNum) THEN DO:
    FIND Customer OF Order NO-LOCK.
    DISPLAY Order.CustNum Customer.Name.
  END.
  DISPLAY Order.OrderNum Order.OrderDate.
END.
```

The following example uses relative positioning to position fields relative to the `CustNum` field.

r-at1.p

```
DEFINE FRAME order-info
  Order.CustNum AT X 50 Y 14
  Customer.Name AT X-OF Order.CustNum + 100 Y 14
  Order.OrderNum AT X-OF Order.CustNum + 225 Y 14
  Order.OrderDate AT X-OF Order.CustNum + 320 Y 14
  WITH TITLE "Order Information" NO-LABELS.

FOR EACH Order NO-LOCK
  BREAK BY Order.CustNum WITH FRAME order-info:
  IF FIRST-OF(Order.CustNnum) THEN DO:
    FIND Customer OF Order NO-LOCK.
    DISPLAY Order.CustNum Customer.Name.
  END.
  DISPLAY Order.OrderNum Order.OrderDate.
END.
```

Notes

- The AT phrase does not left justify the data. It simply specifies the position of the data area. If the data is right justified it may appear to be farther right than you expect.
- If you position a child frame completely outside the virtual area of its parent frame, the AVM raises ERROR at run time when the frame is realized.
- For SpeedScript, you can position objects by row or column, not by pixels.

See also

DEFINE FRAME statement, FORM statement, Frame phrase

AUDIT-ENABLED function

Determines whether a connected database is audit-enabled.

For information about audit-enabling a database, or creating and activating an audit policy for a database, see *Introduction to Security and Auditing*.

Syntax

```
AUDIT-ENABLED( [ integer-expression | logical-name | alias ] )
```

integer-expression

The sequence number of a connected database to query. For example, AUDIT-ENABLED(1) queries the first database, AUDIT-ENABLED(2) queries the second database, and so on. If you specify a sequence number that does not correspond to a connected database, the AVM returns the Unknown value (?).

logical-name or *alias*

The logical name or alias of a connected database to query. These forms require a quoted character string or a character expression. If you specify a logical name or alias that does not correspond to a connected database, the AVM returns the Unknown value (?).

Notes

- If you specify a connected database, the AVM queries that database and returns TRUE if it is audit-enabled. If you do not specify a database, the AVM queries all connected databases and returns TRUE if any one of the connected databases is audit-enabled.
- You can reference the AUDIT-ENABLED function within a preprocessor &IF expression (such as, &IF AUDIT-ENABLED … &ENDIF). For more information, see the &IF, &THEN, &ELSEIF, &ELSE, and &ENDIF preprocessor directives reference entry.

See also

AUDIT-POLICY system handle

AVAILABLE function

Returns a TRUE value if the record buffer you name contains a record and returns a FALSE value if the record buffer is empty.

When you use the FIND statement or the FOR EACH statement to find a record, the AVM reads that record from the database into a record buffer. This record buffer has the same name as the file used by the FIND or FOR EACH statement, unless you specify otherwise. The CREATE statement creates a new record in a record buffer.

Syntax

```
AVAILABLE record
```

record

The name of the record buffer you want to check.

Note: To access a record in a table defined for multiple databases, you might have to qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

Example

In this procedure, the FIND statement with the NO-ERROR option bypasses the default error checking and does not display the message you get. Because `ItemNum` is unique, you do not have to use the AMBIGUOUS function to pinpoint the cause of a record not being AVAILABLE.

r-avail.p

```
REPEAT:
  PROMPT-FOR Item.ItemNum.
  FIND Item USING ItemNum NO-ERROR.
  IF AVAILABLE Item THEN
    DISPLAY Item.ItemName Item.Price.
  ELSE
    MESSAGE "Not found".
END.
```

See also

AMBIGUOUS function, FIND statement, FOR statement, LOCKED function, NEW function (record buffers)

BASE64-DECODE function

Converts a Base64 character string into a binary value. The result is a MEMPTR containing the binary data.

Syntax

```
BASE64-DECODE ( expression )
```

expression

A CHARACTER or LONGCHAR expression containing the string you want to convert.

Note: The BASE64-ENCODE and BASE64-DECODE functions:

- Use the Base64 Alphabet table as found in RFC 4648.
- Do not enforce line lengths with separators.
- Require use of the '=' padding character.
- Do not implement the Filename Safe Alphabet needed for Base64URL encoding.

Example

Following is an example using the BASE64-DECODE function:

```
DEFINE VARIABLE decdmptr AS MEMPTR   NO-UNDO.
DEFINE VARIABLE decdlngc AS LONGCHAR NO-UNDO.

COPY-LOB FROM FILE "C:\myicons\testencode" TO decdlngc.
decdmptr = BASE64-DECODE(decdlngc).
COPY-LOB FROM decdmptr TO FILE "C:\myicons\test.ico".
```

BASE64-ENCODE function

Converts binary data into a Base64 character string, and returns a LONGCHAR containing the character data. The resulting LONGCHAR is in the code page specified by `-cpinternal`.

Syntax

```
BASE64-ENCODE ( expression )
```

expression

A MEMPTR or RAW expression containing the binary data you want to convert.

Note: The `BASE64-ENCODE` and `BASE64-DECODE` functions:

- Use the Base64 Alphabet table as found in RFC 4648.
- Do not enforce line lengths with separators.
- Require use of the '=' padding character.
- Do not implement the Filename Safe Alphabet needed for Base64URL encoding.

Example

Following is an example using the BASE64-ENCODE function:

```
DEFINE VARIABLE encdmptr AS MEMPTR   NO-UNDO.
DEFINE VARIABLE encdlngc AS LONGCHAR NO-UNDO.

COPY-LOB FROM FILE "C:\myicons\test.ico" TO encdmptr.
encdlngc = BASE64-ENCODE(encdmptr).
COPY-LOB FROM encdlngc TO FILE "C:\myicons\testencode".
```

The following example shows encoded strings, with and without padding characters at the end of them:

```
define variable cleartext as longchar no-undo.
define variable encoded as longchar no-undo.
define variable rawdata as memptr no-undo.

// when encoded this string will require padding characters.
cleartext = "needs padding".
copy-lob from cleartext to rawdata convert target codepage "utf-8".
encoded =  base64-encode(rawdata).
message string(encoded) view-as alert-box.


// when encoded, this string does not have padding character at the end of it.
cleartext = ".no padding.".
copy-lob from cleartext to rawdata convert target codepage "utf-8".
encoded =  base64-encode(rawdata).
message string(encoded) view-as alert-box.
```

BEGINS operator

Tests a character expression to see if that expression begins with a second character expression.

Syntax

```
expression1 BEGINS expression2
```

expression1

An expression that has a CHARACTER or LONGCHAR value that you test to see if it begins with *expression2*.

expression2

An expression that has a character value that you want to compare to the beginning of *expression1*. If you specify a null value ("") for *expression2,* the AVM returns all the records in the database.

Examples

In this procedure, the user supplies a customer name or the first characters of a customer name. The procedure finds customer records where the name field begins with the user's input. If the customer file is indexed on the name field, this procedure is very efficient and retrieves only the selected records.

r-bgns.p

```
DEFINE VARIABLE cName NO-UNDO LIKE customer.name LABEL "Name".

REPEAT:
  SET cName WITH SIDE-LABELS.
  FOR EACH Customer NO-LOCK WHERE Customer.Name BEGINS cName:
    DISPLAY Customer.Name Customer.Address Customer.City Customer.State
      Customer.PostalCode.
  END.
END.
```

The next procedure lists exactly the same customers. However, it is much less efficient because it retrieves and examines all customer records, and only displays the ones with the appropriate names.

r-bgns2.p

```
DEFINE VARIABLE cName NO-UNDO LIKE customer.name LABEL "Name".

REPEAT:
  SET cName WITH SIDE-LABELS.
  /* Create MATCHES pattern */
  cName = cName + "*".
  FOR EACH Customer NO-LOCK WHERE Customer.Name MATCHES cName:
    DISPLAY Customer.Name Customer.Address Customer.City Customer.State
      Customer.PostalCode.
  END.
END.
```

Notes

- The BEGINS operator is double-byte enabled. You can use the BEGINS operator to compare strings containing double-byte characters.
- When you use the BEGINS operator to compare LONGCHAR fields, BEGINS always uses the `-cpcoll` collation.
- BEGINS is useful in a WHERE phrase that specifies which records should be retrieved in a FOR EACH block. Unlike the MATCHES operator, which requires that all records in the file be scanned, BEGINS uses an index wherever possible.
- Most character comparisons are case insensitive in ABL. By default, all characters are converted to uppercase prior to comparisons. However, you can define fields and variables as case sensitive (use if strict ANSI SQL adherence is required). If either of the character expressions passed to BEGINS is a field or variable defined as case sensitive, the comparison is case sensitive. In a case-sensitive comparison "SMITH" does not equal "Smith".
- ABL considers trailing blanks in the BEGINS operator. For example, this statement is FALSE:

```
"x" BEGINS "x      "
```

  This is different than comparisons, where trailing blanks are ignored. For example, this statement IS TRUE:

```
"x" = "x     "
```

See also

MATCHES operator

BELL statement

Causes the terminal to make a beep sound.

Note: Does not apply to SpeedScript programming.

Syntax

```
BELL
```

Example

The following procedure dynamically determines the output file to use for a report that lists all customer records. The SET statement gets the name of a file from the user. The SEARCH function returns an unqualified file name if that file already exists in your working directory. If the file exists in your working directory, it displays messages, undoes the work done in the DO block, and lets the user enter another file name. (The procedure determines whether the file is in your working directory. If SEARCH returns a directory other than your current working directory, you receive no messages and it does not undo your work.) After you type a file name that does not already exist, the OUTPUT TO statement directs the output of the procedure to that file.

r-bell.p

```
DEFINE VARIABLE outfile AS CHARACTER NO-UNDO FORMAT "x(8)"
  LABEL "Output file name".

getfile:
DO ON ERROR UNDO, RETRY:
  SET outfile WITH SIDE-LABELS.
  IF SEARCH(outfile) = outfile THEN DO:
    MESSAGE "A file named" outfile "already exists in your directory".
    MESSAGE "Please use another name".
    BELL.
    UNDO getfile, RETRY getfile.
  END.
END.

OUTPUT TO VALUE(outfile).
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.Name Customer.CreditLimit.
END.
```

Note

If the terminal is not the current output device, BELL has no effect.

BLOCK-LEVEL ON ERROR UNDO, THROW statement

Use this statement in a procedure (.p) or class (.cls) file to change the default `ON ERROR` directive to `ON ERROR UNDO, THROW` for all blocks that have a default error directive associated with them. (A simple DO block, for example, does not have default error handling and is not affected by this statement.). This is specifically for `ERROR`, not `STOP` conditions because `STOP` conditions are already thrown by default. The statement must come before any executable or `DEFINE` statements in a file. However, it can come either before or after a `USING` statement.

Syntax

```
BLOCK-LEVEL ON ERROR UNDO, THROW.
```

This statement affects the following block types:

- Main block of an external procedure (.p)
- Internal procedures
- User-defined functions
- Methods of a class
- Class constructors
- Property accessors
- `ON` blocks used as database triggers with `CREATE`, `DELETE`, `WRITE` or `ASSIGN` events
- `REPEAT` blocks
- `FOR` blocks

- DO TRANSACTION blocks

This statement does not affect:

- Destructors
- Error directives that are explicitly coded in individual, non routine-level blocks
- ON blocks that are UI triggers.

Note these alternatives to the BLOCK-LEVEL ON ERROR UNDO, THROW statement:

- Instead of adding the statement to source-code files, you can use the -undothrow 2 startup parameter to change the default error-handling to UNDO, THROW on every block affected by the BLOCK-LEVEL statement during compilation. See the *Startup Command and Parameter Reference* for more information.
- The ROUTINE-LEVEL ON ERROR UNDO, THROW statement can be used if you want to change the default error-handling only on routine-level blocks. (You can use the -undothrow 1 startup parameter to change the default error handling on routine-level blocks to UNDO, THROW during compilation.)

Example

An error propagates from the DO TRANSACTION block to the internal procedure, from the internal procedure up to the main .p file (b-BLOCK-LEVEL-01.p), and finally up to the CATCH block in the calling .p file (b-BLOCK-LEVEL-02.p). An error is raised at each of these levels. At each level the ON ERROR UNDO, THROW directive takes effect.

b-BLOCK-LEVEL-01.p

```
BLOCK-LEVEL ON ERROR UNDO, THROW.

/* Update first order for a customer shipped by a certain date */
DEFINE INPUT PARAMETER custName AS CHAR.
DEFINE INPUT PARAMETER latestShipDate AS DATE.

RUN updateCustomers.

PROCEDURE updateCustomers:
    DO TRANSACTION:
        FIND customer WHERE customer.NAME = custName.
        FIND order OF customer WHERE shipDate < latestShipDate.
        UPDATE order.
    END.
END.
```

b-BLOCK-LEVEL-02.p

```
RUN b-BLOCK-LEVEL-01.p (INPUT cValue, INPUT dDate).

CATCH err AS Progress.Lang.Error:
    MESSAGE "Customer or order not found " err:GetMessage(1)
        VIEW-AS ALERT-BOX INFO BUTTONS OK.
END.
```

Notes

- The BLOCK-LEVEL ON ERROR UNDO, THROW statement guarantees that all unhandled errors in affected blocks are propagated up to the caller. You can decide whether to handle errors at the block level with a CATCH statement, or to handle all errors with a CATCH block at a higher level.

- The BLOCK-LEVEL statement affects the same blocks as the ROUTINE-LEVEL ON ERROR UNDO, THROW statement, plus REPEAT, FOR, and DO TRANSACTION blocks. Therefore, there is no need to add both statements to a file. If both statements exist in a file, the more inclusive BLOCK-LEVEL statement takes precedence.
- Error objects can be thrown from an application server and handled by a CATCH block on an ABL client. To be throwable from an application server to an ABL client, user-defined error classes must be defined on both the server and client sides, and the classes must be defined as SERIALIZABLE. For the full list of restrictions on class-based objects that are passed between application server and client, see the Parameter passing syntax entry. For more information on error handling in general, see *ABL Error Handling*.

See also

ON ENDKEY phrase, ON ERROR phrase, ON QUIT phrase, RETURN statement, RETURN-VALUE function, ROUTINE-LEVEL ON ERROR UNDO, THROW statement, STOP statement, UNDO-THROW-SCOPE attribute

---

BOX function
(Windows only; GUI for .NET only)

Returns an object reference to a .NET System.Object that contains (*boxes*) a .NET mapping of an ABL value. At run time, this mapping depends on the kind of ABL value passed to the function. If you pass an ABL primitive value, the function returns a corresponding .NET mapped object type. If you pass an ABL array of .NET-compatible elements, the function returns a corresponding .NET array object type.

Syntax

```
BOX ( ABL-expression [ , AS-data-type-expression ] )
```

ABL-expression

Specifies an expression with a value in one of the following .NET-compatible ABL data types:

- CHARACTER
- DATE
- DATETIME
- DATETIME-TZ
- DECIMAL
- INT64
- INTEGER
- LOGICAL
- LONGCHAR
- A .NET-compatible ABL array

A .NET-compatible ABL array is an ABL array of one of the listed ABL primitive types or an ABL array of any supported .NET object type (a .NET object type not mapped as specified in Table 24: Implicit mappings between .NET and ABL data types).

AS-data-type-expression

If *ABL-expression* is an ABL primitive type, this is a character expression equal to a keyword (*AS data type*) that matches the explicit .NET mapped data type into which you want to box the specified *ABL-expression*. If *ABL-expression* is an ABL primitive array, the character expression is an AS data type that specifies the explicit .NET mapped data type that the elements of the array should be converted to in the resulting .NET array object. This AS data type must correspond to one of the .NET types that maps appropriately to the ABL primitive type (or array element type) of *ABL-expression* (see Table 25: Explicit mappings between ABL and .NET data types).

If you do not specify *AS-data-type-expression* and *ABL-expression* is an ABL primitive (or primitive array), ABL boxes *ABL-expression* using the default matching .NET type that corresponds to the ABL primitive type (or array element type) of *ABL-expression*.

This option has no meaning when *ABL-expression* is an ABL array of a supported .NET object type.

Example

The following code creates a .NET `DataTable` with a single `DataRow` containing two columns. It then adds data to the row for the two columns. The `Item` indexed property used to access the columns has the data type `System.Object`. So, to add data to each column, you might have to use the BOX function to box the specified ABL value into an appropriate `System.Object` instance for the column. For example:

```
USING System.Data.* FROM ASSEMBLY.

DEFINE VARIABLE dataTable1 AS DataTable  NO-UNDO.
DEFINE VARIABLE dcCustNum  AS DataColumn NO-UNDO.
DEFINE VARIABLE dcName     AS DataColumn NO-UNDO.
DEFINE VARIABLE dcBusType  AS DataColumn NO-UNDO.
DEFINE VARIABLE row1       AS DataRow    NO-UNDO.

dataTable1 = NEW DataTable(INPUT "Customer").

/* Create columns for a dataTable */
dcCustNum = NEW DataColumn(INPUT "CustNum").
dcName    = NEW DataColumn(INPUT "Name").
dcBusType = NEW DataColumn(INPUT "BusType").
dataTable1:Columns:Add(INPUT dcCustNum).
dataTable1:Columns:Add(INPUT dcName).
dataTable1:Columns:Add(INPUT dcBusType).

/* Create a new row */
row1 = dataTable1:NewRow( ).

/* Add data to row */
row1:Item["CustNum"] = 1.
row1:Item["Name"] = "Mr Jones".
    /* Without BOX, this automatically boxes System.Int32 value. */
row1:Item["BusType"] = BOX( 236, "UNSIGNED-BYTE"). /* System.Byte value */
```

Notes

- For many direct assignments of a `System.Object` to an ABL primitive value or .NET-compatible ABL array, use of the BOX function is optional, because ABL automatically boxes the assigned ABL value into its default matching .NET object type. However, one such assignment for which you must use the BOX function is when you want to box an ABL primitive value (or primitive array) as a valid .NET mapped data type (or array of mapped types) other than the default match, for example, when boxing an ABL INTEGER (or INTEGER EXTENT) as a .NET `System.Byte` (or `"System.Byte[]"`).

- If you pass a compatible ABL value or array to an INPUT parameter of a **.NET** method, ABL automatically boxes the ABL value into the matching .NET `System.Object` or array object. For an ABL primitive (or primitive array) value, this automatic boxing also allows you to explicitly specify the .NET data type mapping if you use the AS data type option on the ABL argument that you pass to the INPUT parameter (see the Parameter passing syntax reference entry). This automatic boxing does **not** occur for an ABL method, procedure, or user-defined function passing the same parameters. In this case, you can do an initial direct assignment or use the BOX function to explicitly do the necessary conversion.

See also

Data types, UNBOX function

BUFFER-COMPARE statement

Performs a bulk comparison of two records (source and target) by comparing source and target fields of the same name for equality and storing the result in a field. You can specify a list of fields to exclude, or a list of fields to include. You can also specify `WHEN...THEN` phrases. For all such phrases you specify, the AVM evaluates the `WHEN` portion, and if it evaluates to TRUE, the AVM executes the `THEN` portion.

Syntax

```
BUFFER-COMPARE source
  [ { EXCEPT | USING } field ... ] TO target
  [ CASE-SENSITIVE | BINARY ]
  [ SAVE [ RESULT IN ] result-field ]
  [ [ EXPLICIT ] COMPARES ]:
     [ WHEN field compare-operator expression
       THEN statement-or-block ] ...
  [ END [ COMPARES ] ] [ NO-LOBS ] [ NO-ERROR ]
```

source

The source database table, buffer, temp-table, or work table.

EXCEPT *field*

A list of source fields to exclude from the bulk compare.

USING *field*

A list of source fields to include in the bulk compare. The `USING` option is a positive version of the `EXCEPT` option.

TO *target*

The target database table, buffer, temp-table, or work table.

CASE-SENSITIVE

Directs the AVM to perform a case-sensitive comparison.

BINARY

Directs the AVM to perform a binary comparison.

SAVE RESULT IN *result-field*

A variable or field to contain the result of the comparison. The variable or field must be CHARACTER or LOGICAL.

If *result-field* is CHARACTER, the result is a comma-separated list of fields that failed the comparison, sorted in ascending order.

If *result-field* is LOGICAL, the result is YES if all fields are equal, or NO if any fields are unequal. In either case, BUFFER-COMPARE stops comparing when it encounters the first inequality.

EXPLICIT COMPARES

Opens a block of WHEN options. If you open the block, you must close it with END COMPARES.

WHEN *field*

Any data field in the source.

BUFFER-COMPARE removes this field from a USING list or adds this field to an EXCEPT list. This removes the field from the bulk compare and from result-field.

compare-operator

Represents one of the following: LT, LE, GT, GE, EQ, NE, MATCHES, BEGINS, or CONTAINS.

Note: For BLOB or CLOB fields, you can only use the EQ (=) or NE (<>) operators.

expression

Any valid ABL expression.

THEN *statement-or-block*

Any ABL statement or block. The statement or block executes when the WHEN clause evaluates to TRUE.

END COMPARES

Closes the block of WHEN phrases.

NO-LOBS

Directs the AVM to ignore large object data when comparing records that contain BLOB or CLOB fields.

Caution: When using this option, you get no warning if a LOB field has changed. This can lead to inappropriate results. Therefore, before using this option, you must understand the nature of your data and be sure that logic using this option will not result in inconsistent or out-of-date data in the database.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Notes

- At compile time, BUFFER-COMPARE:
  - Fails to compile if any source-target field pair is not type compatible. An example of such a pair is a field that is LOGICAL in the source, but DECIMAL in the target.
  - Excludes from the bulk comparison all EXCEPT *field* fields and all WHEN *field* fields.
  - Automatically excludes from the bulk comparison fields that appear in the source but not in the target.
  - Tries to bind unqualified field names that appear in the EXCEPT and USING options to the source buffer.
- At run time, BUFFER-COMPARE:
  - Compares all fields not in the EXCEPT phrase and all fields not in the WHEN phrase for equality
  - Stores the result in the field that the SAVE phrase specifies, if any
  - Evaluates each WHEN option, executing it if its condition evaluates to TRUE

Note: This behavior is different from the behavior of the ABL CASE statement, which executes only the first WHEN option whose condition evaluates to TRUE.

- When comparing records that contain BLOB fields, the AVM performs a binary comparison on the BLOB data associated with the source and target records, and reports the results of the comparison.
- You cannot use the BUFFER-COMPARE statement to compare records that contain CLOB fields, unless one or both of the corresponding fields contain the Unknown value (?). However, you can convert CLOB fields to LONGCHAR values and use the EQ, GE, GT, LE, LT, or NE comparison operator to compare the LONGCHAR values.
- The statement supports three types of comparison for CLOB fields, namely, case-sensitive, binary and case-neutral.
- Use the NO-LOBS option with the BUFFER-COMPARE statement to ignore large object data when comparing records that contain BLOB or CLOB fields. You can also use the EXCEPT option to exclude BLOB and CLOB fields from the compare.

See also

BUFFER-COPY statement , NO-ERROR option

BUFFER-COPY statement

Performs a bulk copy of a source record to a target record by copying each source field to the target field of the same name. You can specify a list of fields to exclude from the bulk copy, or a list of fields to include in the bulk copy.

Syntax

```
BUFFER-COPY source [ { EXCEPT | USING } field ... ]
  TO target [ ASSIGN assign-expression ... ] [ NO-LOBS ] [ NO-ERROR ]
```

source

The source database table, buffer, temp-table, or work table.

EXCEPT *field* ...

A list of space-separated source fields to exclude from the bulk copy.

USING *field* ...

A list of space-separated source fields to include in the bulk copy. The USING option is simply a positive version of the EXCEPT option.

TO *target*

The source database table, buffer, temp-table, or work table.

ASSIGN *assign-expression*

A space-separated list of any valid ABL ASSIGN statements (without the EXCEPT option, which BUFFER-COPY already provides). BUFFER-COPY performs each *assign-expression* and automatically excludes the field on the left side ("destination") of each *assign-expression* from the bulk copy-except for field extents (subscripted fields). If a field extent appears on the left side of an *assign-expression*, BUFFER-COPY does not automatically exclude that extent (such as customer.mnth-sales[1]) or the field as a whole (such as customer.mnth-sales) from the bulk copy.

NO-LOBS

Directs the AVM to ignore large object data when copying records that contain BLOB or CLOB fields.

Caution: Using this option can create the potential for errors in your data and lead to inappropriate results. Therefore, before using this option, you must understand the nature of your data and be sure that logic using this option will not result in inconsistent or out-of-date data in the database.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Note: Compound assignment operators (+=, -=, *=, /=) can be used with the BUFFER-COPY statement. For more information, see += Addition assignment operator, -= Subtraction assignment operator, *= Multiplication assignment operator, and /= Division assignment operator.

Notes

- At compile time, BUFFER-COPY:
  - Fails to compile if any source-target field pair is not type compatible

  - Excludes from the bulk copy all EXCEPT *field* fields, and all *assign-expression* fields on the left side of the assignment
  - Automatically excludes fields that appear in the source but not the target from the bulk copy
  - Tries to bind unqualified field names that appear in the EXCEPT and USING options to the source buffer
- At run time, BUFFER-COPY:
  - Creates a target record if none already exists and executes any applicable CREATE triggers
  - Assigns all matching fields that do not appear in the EXCEPT or ASSIGN options
  - Performs each *assign-expression* in the ASSIGN option, one-by-one
- The BUFFER-COPY statement, like the VALIDATE statement, must appear within the scope of a FIND, a FOR EACH, or a CREATE statement that references the source table.
- If a BUFFER-COPY statement references a target buffer for the first time, ABL regards this reference as a "free reference" and scopes the buffer to the nearest enclosing block that can scope records. For more information on free references, see the topics on block properties in *Develop ABL Applications*.
- With respect to transaction processing, ABL treats a BUFFER-COPY statement the same way it would treat equivalent ASSIGN statements. For more information on transaction processing, see the topics on transactions in *Develop ABL Applications*.
- The compiler's XREF facility automatically creates a REFERENCE for each field in the fields list, a TABLE-REFERENCE for the source and target buffers, ACCESS and UPDATE references for any fields in the ASSIGN option, and ACCESS (or UPDATE) references for each source (or target) field that participates in the bulk copy.
- When copying records that contain a BLOB or CLOB field, the AVM copies the object data associated with the source record to the target record. If the BLOB or CLOB field in the source record contains the Unknown value (?), the AVM stores the Unknown value (?) in the BLOB or CLOB field of the target record. If the target record already has object data associated with it, the AVM deletes that object data before copying the new object data.
- Use the NO-LOBS option with the BUFFER-COPY statement to ignore large object data when copying records that contain BLOB or CLOB fields. More specifically:
  - When you copy a source record to a new target record, the AVM sets the value of the BLOB or CLOB field in the target record to the Unknown value (?).
  - When you copy a source record to an existing target record, the AVM does not change the value of the BLOB or CLOB field in the existing target record.

  You can also use the EXCEPT option to exclude BLOB and CLOB fields from the copy.

See also

BUFFER-COMPARE statement , NO-ERROR option

BUFFER-GROUP-ID function

Returns the group ID (as an integer) of the tenant group to which the current record in a specified record buffer belongs. If the buffer does not contain a record from a tenant group, the function returns the Unknown value (?).

Syntax

```
BUFFER-GROUP-ID ( buffer-name )
```

buffer-name

An identifier that specifies the name of a record buffer.

Note

BUFFER-GROUP-ID can be used in a WHERE or TENANT-WHERE option as long as the buffer specified by *buffer-name* is not the same as the buffer of the query or FOR EACH statement.

See also

BUFFER-GROUP-ID attribute, BUFFER-GROUP-NAME function

BUFFER-GROUP-NAME function

Returns the name (as a character string) of the tenant group to which the current record in a specified record buffer belongs. If the buffer does not contain a record from a tenant group, the function returns the Unknown value (?).

Syntax

```
BUFFER-GROUP-NAME ( buffer-name )
```

buffer-name

An identifier that specifies the name of a record buffer.

Note

BUFFER-GROUP-NAME can be used in a WHERE or TENANT-WHERE option as long as the buffer specified by *buffer-name* is not the same as the buffer of the query or FOR EACH statement.

See also

BUFFER-GROUP-ID function, BUFFER-GROUP-NAME attribute

BUFFER-PARTITION-ID function

Returns the partition ID (as an integer) of the partition of the current record in a specified buffer.

Syntax

```
BUFFER-PARTITION-ID ( buffer-name )
```

buffer-name

An identifier that specifies the name of a record buffer.

If the buffer is not populated with a record, this function returns the Unknown value (?). If there is a record in the buffer but it is not for a partitioned table, this function returns zero (0).

Notes

- If you use this function in a WHERE clause, it will be evaluated for each iteration on the client side after the record has been retrieved. It will not participate in any index bracketing, so unless there are other partition-column criteria in the same WHERE clause, it will be very inefficient.
- The partition ID is only unique within a given table.

See also

BUFFER-PARTITION-ID attribute, IS-PARTITIONED attribute

BUFFER-TENANT-ID function

Returns the tenant ID (as an integer) of the tenant that owns the current record in a specified buffer.

Syntax

```
BUFFER-TENANT-ID ( buffer-name )
```

buffer-name

An identifier that specifies the name of a record buffer.

If the buffer is not populated with a record, this function returns the Unknown value (?). If the buffer is for a shared table, a temp-table, or for the default tenant of a multi-tenant table, the function returns the value zero (0).

If the buffer contains a record for a tenant group and the user is a super tenant, the function returns the tenant ID of one of the members of the group. The tenant ID returned is indeterminate, but is always the tenant ID of a tenant in the group.

For a regular tenant, the record in the buffer always belongs to that tenant. So, the function always returns the user's own tenant ID. This is true even if the record belongs to a tenant group.

Notes

- BUFFER-TENANT-ID can be used in a WHERE or TENANT-WHERE option as long as the buffer specified by *buffer-name* is not the same as the buffer of the query or FOR EACH statement.
- This function can be used by database triggers on the buffers passed to the trigger to get the tenant ID of these buffers. This allows the application provider to code tenant-specific database triggers.

See also

BUFFER-TENANT-ID attribute, BUFFER-TENANT-NAME function

BUFFER-TENANT-NAME function

Returns the name (as a character string) of the tenant that owns the current record in a specified buffer.

Syntax

```
BUFFER-TENANT-NAME ( buffer-name )
```

buffer-name

An identifier that specifies the name of a record buffer.

If the buffer is not populated with a record, this function returns the Unknown value (?). If the buffer is for a shared table or a temp-table, the function returns the empty string ("").

If the buffer contains a record that belongs to a tenant group and the user is a super tenant, the function returns the tenant name of one of the members of the group. The tenant name returned is indeterminate, but is always the name of a tenant in the group.

For a regular tenant user, the record in the buffer always belongs to that tenant. So, the function always returns the user's own tenant name. This is true even if the record belongs to a tenant group.

Notes

BUFFER-TENANT-NAME can be used in a WHERE or TENANT-WHERE option as long as the buffer specified by *buffer-name* is not the same as the buffer of the query or FOR EACH statement.

See also

BUFFER-TENANT-ID function, BUFFER-TENANT-NAME attribute

CALL statement

Transfers control to a dispatch routine (PRODSP) that then calls a C function. You write the C function using ABL Host Language Call (HLC) interface.

ABL HLC consists of a collection of C functions that:

- Obtain data from ABL shared variables and buffers
- Set data in ABL shared variables and buffers
- Control screen modes
- Provide ABL-like messages in the message area at the bottom of the screen

Using HLC, you can extend ABL with your own C functions.

Syntax

```
CALL routine-identifier [ argument ] ...
```

routine-identifier

The name the PRODSP dispatch routine used to identify the C function to call.

argument

One or more arguments that you want to pass to the C function.

See also

RUN statement

CAN-DO function

Checks a user ID against a list of one or more user ID matching patterns that can be used to indicate what users have access to a given application function. The function returns TRUE if the specified user ID has access according to the list. Thus, you can implement run-time authorization checking for any procedure or class in your application.

Syntax

```
CAN-DO ( id-pattern-list [ , userid ] )
```

id-pattern-list

A constant, field name, variable name, or expression that evaluates to a list of one or more user ID patterns. Each user ID pattern must be in the form of a table and field permission string that you might specify when editing data security with database administration tools. If the expression contains multiple user ID patterns, you must separate the patterns with commas. There can be no embedded spaces within or between the patterns in the list.

The following table lists some basic user ID patterns that you can include in *id-pattern-list*, where *user-ID* can be a fully qualified or non-qualified user ID, depending on your application and authentication design, and *string* is a sequence of characters from any user ID. For more information on fully qualified and non-qualified user ID's, see the reference entries for the QUALIFIED-USER-ID attribute, USER-ID attribute, and USERID function.

The patterns shown also support the use of a period (.) as a wildcard for a single character matches. For example, `CAN-DO("a.c","abc")` evaluates to TRUE.

Table 10: Basic patterns to use in user ID pattern lists

Value	Meaning
*	All users have access.

Value	Meaning
user-ID	This user has access.
!*user-ID*	This user does not have access.
*string**	Users whose IDs begin with *string* have access.
!*string**	Users whose IDs begin with *string* do not have access.

You can use any combination of patterns to define *id-pattern-list*. These patterns are fully consistent with the permissions that you can define for OpenEdge database tables and fields using database administration tools.

The following table shows some examples of fully qualified user ID matches.

Table 11: Fully qualified user ID matches

This pattern...	Matches...
	The blank user name in the blank domain
*	All user ID's
mark@acme.com	The "mark" user name in the "acme.com" domain
mark	The "mark" user name in the blank domain
mark*	All user names that begin with "mark" in the blank domain
*jones	All user names that end with "jones" in the blank domain
db*user	All user names that begin with "db" and end with "user" in the blank domain
@acme	The blank user name in the "acme" domain
*@acme	All user names in the "acme" domain
@	The blank user name in the blank domain
mark@*	The "mark" user name in any domain
@	Any user name in any domain
mark*@acme.*	Any user name that begins with "mark" in any domain that begins with "acme."
@.admins	Any user name in any domain that ends with ".admins"

Note: For a more complete description of the possible patterns and their meanings, see *Introduction to Identity Management* and the Database Administration online help.

userid

A character expression that evaluates to a user ID value that can be returned from the QUALIFIED-USER-ID attribute, USER-ID attribute, or USERID function. The *userid* is checked against *id-pattern-list*. If you do not enter *userid*, the compiler inserts the USERID function by default, which is evaluated each time you run the procedure or class. If the compiler inserts the USERID function, the function does not reference a database name and therefore only works for an ABL session with one database connection. If you explicitly use the USERID function and have more than one database connected, be sure to include the database name with the function call, for example, USERID("Sports2020").

Examples

The r-cando.p procedure is based on an activity permission table called permission. This permission table is not included in any of the installed sample databases. However, the records in this table contain two fields and might look something like the following:

Activity	Can-Run
custedit	manager,salesrep
ordedit	manager,salesrep
itemedit	manager,inventory
reports	manager,inventory,salesrep

In r-cando.p the FIND statement reads the record for the activity "custedit" in the permission table. The CAN-DO function compares the value of USERID (the user ID of the single session database connection) with the list of users in the Can-Run field of the custedit record. If the user ID is "manager" or "salesrep", the procedure continues executing. Otherwise, the procedure displays a message and control returns to the calling procedure.

r-cando.p

```
DO FOR permission:
  FIND permission WHERE Activity = "custedit".
  IF NOT CAN-DO(permission.Can-Run, USERID) THEN DO:
    MESSAGE "You are not authorized to run this procedure".
    RETURN.
  END.
END.
```

In this next example, the CAN-DO function compares the value of USERID against the list of user ID patterns passed as a constant string to the function. The values in the constant string include "manager" and any user IDs beginning with "acctg" except "acctg8". If there is no match between the two values, the procedure displays a message and then exits.

r-cando2.p

```
IF NOT CAN-DO("manager,!acctg8,acctg@*", USERID) THEN DO:
  MESSAGE "You are not authorized to run this procedure.".
  RETURN.
END.
```

Notes

- If *id-pattern-list* contains contradictory values, the first occurrence of a value in the list applies. For example, CAN-DO("abc,!abc*") evaluates to TRUE, because the user ID, "abc", appears before "!abc" in *id-pattern-list*.
- If *id-pattern-list* is exhausted without a match, CAN-DO returns a value of FALSE. Therefore, "!abc" restricts "abc" and everyone else, including the blank user ID (""). To restrict "abc" only and allow everyone else, use "!abc,*".
- A *userid* comparison against *id-pattern-list* is not case sensitive.
- If a user is logged into a UNIX system as root, the AVM allows access to the procedure even if access is denied by the *id-pattern-list.* You must specifically deny root access by adding "!root" to the *id-pattern-list*.
- In addition to the examples shown above, you can use the CAN-DO function to compare a *userid* other than that of the current user against the list of values in *id-pattern-list*. For example, to assign a department *userid* to users "smith" and "jones" when they start the ABL session, you can prompt these users for a department *userid* and *password* and check them against a table of department ID's.
- You can pass user IDs to the CAN-DO function, other than the default, that have been set using command-line database connections, the CONNECT statement, the SECURITY-POLICY:SET-CLIENT( ) method, the SET-DB-CLIENT function, or the SETUSERID function.
- You can use the CAN-DO function to match a user ID against run-time table and field permissions stored in an OpenEdge RDBMS by accessing the user ID patterns stored in the _Can-* fields of the _File metaschema table.
- ABL raises an error if you omit *userid* and one of the following conditions exists:
  - There is no database connected.
  - More than one database is currently connected.
- By default, CAN-DO treats "@" as a delimiter for the domain name in a fully qualified user ID. However, you can change this functionality so that CAN-DO treats "@" as a regular character by using the CAN-DO-DOMAIN-SUPPORT attribute of the SECURITY-POLICY handle or the -nocandodomain startup parameter. For more information on the -nocandodomain startup parameter, see the *Startup Command and Parameter Reference*.
- Using the # character as an argument to the CAN-DO function may lead to incorrect results. # should only be used to distinguish GRANTable versus non-GRANTable permissions in SQL89.

See also

CAN-DO-DOMAIN-SUPPORT attribute, CONNECT statement, QUALIFIED-USER-ID attribute, SET-CLIENT( ) method, SET-DB-CLIENT function, SETUSERID function, USERID function

CAN-FIND function

Returns a TRUE value if a record is found that meets the specified FIND criteria; otherwise it returns FALSE. CAN-FIND does not make the record available to the procedure. You typically use the CAN-FIND function within a VALIDATE option in a data handling statement, such as the UPDATE statement.

You can use CAN-FIND to see if a record exists with less system overhead than that of a FIND statement. The query capabilities are similar. CAN-FIND is also useful for implementing inner joins among database tables.

Syntax

```
CAN-FIND
  (
    [ FIRST | LAST ] record [ constant ]
    [ OF table ] [ WHERE expression ] [ USE-INDEX index ]
    [ USING [ FRAME frame ] field
        [ AND [ FRAME frame ] field ] ...
    ]
    [ SHARE-LOCK | NO-LOCK ] [ NO-WAIT ] [ NO-PREFETCH ]
  )
```

You can specify the OF, WHERE, USE-INDEX, and USING options in any order.

FIRST

Returns TRUE if CAN-FIND locates a record that meets the specified criteria; otherwise returns FALSE.

LAST

Returns TRUE if CAN-FIND locates a record that meets the specified criteria; otherwise returns FALSE.

record

The record buffer you are checking for existence.

To use CAN-FIND to locate a record in a table defined for multiple databases, you might have to qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

constant

The table you want to use has a primary index; the *constant* is the value of the last component field of that index for the record you want.

OF *table*

Qualifies the records to use by relating the record to a record in another table.

WHERE *expression*

Qualifies the record that CAN-FIND searches for. The expression must return a TRUE or FALSE value.

USE-INDEX *index*

Identifies the index you want CAN-FIND to use to find a record. If you do not use this argument, the AVM selects an index to use based on the criteria specified with the WHERE, USING, OF, or *constant* arguments.

USING [FRAME *frame*] *field* [AND [FRAME *frame*] *field*]

One or more names of fields you want to use to search for a record. The field you name in this argument must have been previously entered into a screen field, usually with a PROMPT-FOR statement. The field must be viewed as a fill-in or text widget.

SHARE-LOCK

Specifies that CAN-FIND determines whether the record can be SHARE-LOCKed. If you use this option without the NO-WAIT option, and if the record is EXCLUSIVE-LOCKed, CAN-FIND waits until that lock is released before returning. If you use SHARE-LOCK with the NO-WAIT option, then CAN-FIND returns a FALSE value immediately if the record is EXCLUSIVE-LOCKed.

NO-LOCK

Specifies that CAN-FIND determines whether the record can be accessed with the NO-LOCK option. This is the default for CAN-FIND.

NO-WAIT

Causes CAN-FIND to return immediately and return FALSE if the record is locked by another user.

If you use NO-WAIT together with a SHARE-LOCK and the record found is EXCLUSIVE-LOCKed, the CAN-FIND function does not wait and returns FALSE.

NO-PREFETCH

Specifies that only one record can be sent across the network at a time. If you do not specify this option, the AVM might send more than one record from the server to the client in each network packet.

Example

In the following procedure, the UPDATE statement uses the VALIDATE option to make sure that the salesrep entered matches one of the salesreps in the database. The VALIDATE option uses the CAN-FIND function to find the record.

r-canfnd.p

```
REPEAT:
  CREATE Customer.
  UPDATE Customer.CustNum Customer.Name Customer.SalesRep
    VALIDATE(CAN-FIND(SalesRep.SalesRep
      WHERE SalesRep.SalesRep = Customer.SalesRep),
      "Invalid sales rep -- please re-enter").
END.
```

Notes

- Fields do not have to be indexed to use them in a CAN-FIND function. For example, you can use the following CAN-FIND function with the `Sports2020` database, even though the `State` field is not indexed:

  ```
  CAN-FIND(FIRST Customer WHERE Customer.State = "NH")
  ```

  However, when you use CAN-FIND on a non-indexed field, the response might be slow, as with a FIND.

- You can name more than one field as part of the selection criteria. For example, the following CAN-FIND function works with the `Sports2020` database:

```
CAN-FIND(Customer WHERE Customer.CustNum = x AND Customer.Name = y)
```

- CAN-FIND supports selection criteria that uses inequality matches. Therefore, you can use Boolean operations in WHERE clauses.
- EXCLUSIVE lock is not allowed in a CAN-FIND because CAN-FIND does not return a record.
- If you use the CAN-FIND function to find a record in a work table, the AVM disregards the NO-WAIT, SHARE-LOCK, and NO-LOCK options.
- You can nest CAN-FIND functions. For example, you can use CAN-FIND(... WHERE CAN-FIND(...WHERE CAN-FIND, etc.
- The CAN-FIND function does not cause FIND triggers to execute; hence a procedure can use this function to bypass the FIND trigger and check for the existence of records. Anyone writing a FIND trigger for security reasons should be aware of this.
- You cannot use the CAN-FIND function in a query's WHERE clause. Doing so generates a compiler error.
- Within a CAN-FIND function, if you compare tables or fields from multiple databases, you must explicitly specify the database name along with the table and field name.
- CAN-FIND does not raise a run-time error if the buffers in its predicate are unavailable. Developers should check for record availability when using outside buffers for comparison in a CAN-FIND statement and ensure that the buffer contains the expected value prior to using it so that the CAN-FIND function correctly reports the record availability.

See also

FIND statement

CAN-QUERY function

Returns a logical value indicating whether you can query a specified attribute or method for a specified widget.

Syntax

```
CAN-QUERY ( handle , attribute-name )
```

handle

An expression that evaluates to a handle. The handle must refer to a valid widget.

attribute-name

An expression that evaluates to a character-string value. The contents of the string must be an attribute or method name. For more information on attributes, see the Handle Attributes and Methods Reference.

Example

The following example prompts for a widget type and an attribute. It creates a widget of the specified type and passes a handle to that widget and the attribute you specified to the CAN-QUERY and CAN-SET functions. Then it reports whether the attribute can be queried or set for that widget.

r-prog.p

```
DEFINE VARIABLE attribute   AS CHARACTER NO-UNDO FORMAT "x(24)"
  LABEL "Attribute".
DEFINE VARIABLE queryable   AS LOGICAL   NO-UNDO VIEW-AS TOGGLE-BOX
  LABEL "Query".
DEFINE VARIABLE setable     AS LOGICAL   NO-UNDO VIEW-AS TOGGLE-BOX
  LABEL "Set".
DEFINE VARIABLE temp-handle AS HANDLE    NO-UNDO.
DEFINE VARIABLE widget-type AS CHARACTER NO-UNDO FORMAT "x(24)"
  LABEL "Widget".

FORM widget-type attribute setable queryable.

REPEAT:
  UPDATE widget-type attribute.
  CREATE VALUE(widget-type) temp-handle.
  queryable = CAN-QUERY(temp-handle, attribute).
  setable   = CAN-SET(temp-handle, attribute).
  DISPLAY queryable setable.
  DELETE WIDGET temp-handle.
END.
```

Notes

For SpeedScript, use with buffer-field, buffer-object, buffer, and query-object handles.

See also

CAN-SET function, LIST-QUERY-ATTRS function, LIST-SET-ATTRS function

CAN-SET function

Returns a logical value indicating whether you can set a specified attribute for a specified widget.

Syntax

```
CAN-SET ( handle , attribute-name )
```

handle

An expression that evaluates to a handle. The handle must refer to a valid widget.

attribute-name

An expression that evaluates to a character-string value. The contents of the string must be an attribute name. For more information on attributes, see the Handle Attributes and Methods Reference.

Notes

- For a field-level widget, the CAN-SET function always returns TRUE for the FRAME attribute; however, you can set the frame attribute only if the widget is dynamic. Therefore, before setting the FRAME attribute for a widget, you can test that the operation is valid with a statement similar to the following:

```
IF CAN-SET(my-handle, "FRAME") AND my-handle:DYNAMIC THEN
  my-handle:FRAME = frame-handle.
```

- For SpeedScript, use with buffer-field, buffer-object, buffer, and query-object handles.

See also

CAN-QUERY function, LIST-QUERY-ATTRS function, LIST-SET-ATTRS function

---

CAPS function

Converts any lowercase characters in a CHARACTER or LONGCHAR expression to uppercase characters, and returns the result. The data type of the returned value matches the data type of the expression passed to the function.

Syntax

```
CAPS ( expression )
```

expression

A constant, field name, variable name, or expression that results in a CHARACTER or LONGCHAR value.

Example

In the following code example, the CAPS function converts the characters in the state field to uppercase:

r-caps.p

```
REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum.
  UPDATE Customer.Name Customer.Address Customer.City Customer.State.
  Customer.State = CAPS(Customer.State).
  DISPLAY Customer.State.
END.
```

Notes

- The CAPS function returns uppercase characters relative to the settings of the Internal Code Page (`-cpinternal`) and Case Table (`-cpcase`) startup parameters. For more information on these parameters, see *Startup Command and Parameter Reference*.

- The CAPS function is double-byte enabled. The specified expression can yield a string containing double-byte characters; however, the CAPS function changes only single-byte characters in the string.

See also

LC function

CASE statement

Provides a multi-branch decision based on the value of a single expression.

Syntax

```
CASE expression :
  { WHEN value [ OR WHEN value ] ... THEN
      { block | statement }
  } ...
  [ OTHERWISE
      { block | statement }
  ]
END [ CASE ]
```

expression

The expression that determines which branch of code to execute. The *expression* parameter can be any valid ABL expression. It can include comparisons, logical operations, and parentheses.

WHEN *value* [OR WHEN *value*] . . . THEN

Each *value* is an expression that evaluates to a possible value for *expression*. If *value* matches the current value of *expression*, then the associated block or statement executes.

OTHERWISE

Introduces a block or statement to execute when the value of *expression* does not match any *value* in any of the WHEN clauses.

block

A DO, FOR, or REPEAT block. If you do not use a block, then you can only use a single statement for the WHEN or OTHERWISE clause.

statement

A single ABL statement. If you want to use more than one statement, you must enclose them in a DO, FOR, or REPEAT block.

END [CASE]

Indicates the end of the CASE statement. You can include the CASE keyword here to improve readability; it has no effect on the code.

Example

The following fragment shows a simple example of a CASE statement:

r-case.p

```
DEFINE VARIABLE pay-stat AS INTEGER NO-UNDO INITIAL 1.

UPDATE pay-stat VIEW-AS RADIO-SET
  RADIO-ITEM unpaid   1 LABEL "Unpaid"
  RADIO-ITEM part     2 LABEL "Partially paid"
  RADIO-ITEM paid     3 LABEL "Paid in full".

CASE pay-stat:
  WHEN 1 THEN
    MESSAGE "This account is unpaid.".
  WHEN 2 THEN
    MESSAGE "This account is partially paid.".
  WHEN 3 THEN
    MESSAGE "This account is paid in full.".
END CASE.
```

Notes

- Each *value* must have the same data type as *expression*. If the data types do not match, the compiler reports an error.
- Both *value* and *expression* can be enums, including ABL and .NET enums, but each *value* must be the same enum type as *expression*.
- You can specify any number of WHEN clauses within the CASE statement.
- You can specify only one OTHERWISE clause for a CASE statement. If you use the OTHERWISE clause, it must be the last branch in the statement.
- When a CASE statement is executed, the AVM evaluates *expression* and evaluates each *value* for each branch in order of occurrence until it finds the first *value* that satisfies the condition. At that point the AVM executes that branch and does not evaluate any other *value* for that branch or any other branches. If no matching *value* is found, then the OTHERWISE branch is executed, if given. If the OTHERWISE branch is not given and no matching *value* is found, then no branch of the CASE statement is executed and execution continues with the statement after the CASE statement.
- After a branch of the CASE statement is executed, the AVM leaves the CASE statement and execution continues with the statement following the CASE statement.
- If a LEAVE statement is executed within any branch of a CASE statement, the AVM leaves the closest block (other than a DO block) that encloses the CASE statement.

CAST function

Returns a new object reference to the same class instance as an existing object reference, but with a different data type. This different data type is cast from the object type of the original object reference according to another specified object type. The two object types must be related, where one is a class type and the other is a subclass of that class type or where one is an interface type and the other is a class that implements the interface of that type.

When you cast an object reference, ABL treats it as if it referenced an instance of the object type to which it is cast. The underlying class hierarchy of the object instance does not change.

Syntax

```
CAST( object-reference, object-type-name ).
```

object-reference

An object reference defined with the object type to be cast.

object-type-name

Specifies the type name of an ABL or .NET class or interface type to which the object reference is cast. This object type must be a class type in a class hierarchy that includes or implements the *object-reference* data type or it must be an interface type that the *object-reference* data type implements. Specify the object type name using the syntax described in the Type-name syntax reference entry. With an appropriate USING statement, you can also specify an unqualified class or interface name alone.

Notes

- You typically cast an object reference down a class hierarchy—that is, from a super class to a derived class within a class hierarchy, or from an interface to a class that implements that interface. However, you do not always need to explicitly cast an object reference. Because a derived class contains all the super classes in its inherited class hierarchy, ABL implicitly casts any object reference up within its class hierarchy, and because a class that implements an interface implements all of the methods specified for the interface, ABL implicitly casts any object reference from an implementing class to any interface that the class implements.
- At compile time, ABL verifies that the specified object type is within the class hierarchy of the specified object reference. At run time, the AVM checks the validity of the cast operation. Therefore, if you access a class member on the cast object reference that exists for the cast data type, but the referenced object at run time does not actually define the accessed class member, the AVM raises ERROR at run time.
- A .NET generic type can be part of a cast. For example, you can cast from a `System.Object` to a `"System.Collections.Generic.List<SHORT>"`, because all .NET classes, including generic classes, derive from the .NET root class. However, note that you cannot cast from a `"System.Collections.Generic.List<System.Object>"` to a `"System.Collections.Generic.List<System.Windows.Forms.Button>"`. You cannot assign a `"List<Button>"` reference to an object reference defined as a `"List<Object>"`, because, even though the type parameters are compatible, the two objects as a whole are not equivalent and have no inheritance relationship. Therefore, a cast between these two objects cannot work either. For more information on .NET generic types, see the Data types reference entry.
- You can also use the DYNAMIC-CAST function to cast object references to object types determined at run time. This is especially useful in object-oriented applications that conform to the OpenEdge Reference

Architecture (OERA). For more information on the OERA, see Progress Communities: https://community.progress.com/.

- You can use the CAST function to cast a parameter in a parameter list for a method using the following syntax:

```
method-name( INPUT CAST( object-reference, subclass-name ), ... ).
```

- You can use the CAST function to cast a temp-table field, which is defined as a `Progress.Lang.Object`, to use as an object of another class type. For example:

```
DEFINE VARIABLE rCustObj AS CLASS acme.myObjs.CustObj.

DEFINE TEMP-TABLE mytt FIELD CustObj AS Progress.Lang.Object.

rCustObj = CAST(mytt.CustObj, acme.myObjs.CustObj).
```

  You can now use the object reference in `RCustObj` to invoke methods in the `acme.myObjs.CustObj` class.

- You can use the CAST function to cast an object reference to a subclass and invoke a method defined in that subclass using the following syntax:

```
CAST( object-reference, object-type-name ):method-name( parameters ).
```

  You can also use this syntax to invoke a method on a class that implements the referenced interface from which you cast the specified class.

See also

DYNAMIC-CAST function, Type-name syntax, USING statement

CATCH statement

Defines a `CATCH` block, which allows you to trap an error or stop object and write code to handle it.

A `CATCH` block can be referred to as an *end block* because it defines end-of-block processing for the block that encloses it. End blocks are always part of another block called the *associated block*. End blocks must appear in the associated block after the last executable statement and before the `END` statement. The other type of end block is the `FINALLY` block.

The `CATCH` statement defines the start of an end block that only executes if a condition is raised in its associated block and the object type of the condition raised matches the class specified in the `CATCH` statement (or a subclass of that class). The `CATCH` block thus executes when an object of the specified error or stop type is *caught* by the `CATCH` statement, at which point the object can then be *handled* in the `CATCH` block.

When the condition is raised, if there is an active transaction for the associated block, the transaction is undone before the AVM begins executing the statements within the `CATCH` block.

The `CATCH` block executes once for each iteration of its associated block that raises a compatible error. A block can have multiple `CATCH` blocks, and all must come at the end of the associated block.

There can only be one CATCH block for each specific condition type in a block. However, the block also handles objects for its subtypes. So, it is possible there can be more than one CATCH block that is compatible with a particular condition's type. In this case, the AVM executes the first CATCH block it encounters that is compatible. For this reason, CATCH blocks should be arranged from the most specific type to the most general.

CATCH blocks are covered in greater depth in *ABL Error Handling*.

Syntax

```
CATCH object-variable AS [ CLASS ] condition-class:
    catch-logic
END [ CATCH ] .
```

object-variable

The variable name that references the object caught by this block. It is not necessary to define the *object-variable* ahead of time. The AVM recognizes a new variable name on the CATCH statement as a new *object-variable* definition within the current scope. Each CATCH in an associated block must have a unique *object-variable*. You can reuse an *object-variable* name in a different associated block, if its type is the same as the previous use. For all blocks with their own variable scope, such as object methods or internal procedures, a CATCH statement inside that context may reuse the same variable name as a CATCH statement outside of that context even if the type is different.

condition-class

The code within a CATCH block only executes if a condition of type, *condition-class* (or a subtype), is raised within the body of the associated block.

catch-logic

Any valid ABL statement. This typically contains a MESSAGE statement to display or log an error, or code to handle user-defined application conditions.

Examples

Example 1

In the following example, the CATCH block handles any ABL system error.

```
FIND Customer WHERE CustNum = 5000. /* Will fail */

/* Won't execute because FIND fails */
MESSAGE "Customer found" VIEW-AS ALERT-BOX.

/* The associated block for this CATCH block is the main block of the .p */
CATCH eSysError AS Progress.Lang.SysError:
    MESSAGE "From CATCH block..." SKIP
            eSysError:GetMessage(1)
        VIEW-AS ALERT-BOX.
END CATCH.
```

Example 2

The following example illustrates how *object-variable* names can be reused.

```
DEFINE VARIABLE oneError AS CLASS Progress.Lang.SysError.
    /* This definition is not necessary. */

DO ON ERROR UNDO, LEAVE:
    FIND FIRST Customer WHERE CustNum = 5000.

    CATCH oneError AS Progress.Lang.SysError:
        MESSAGE oneError:GetMessage(1) VIEW-AS ALERT-BOX.
    END CATCH.

    CATCH twoError AS Progress.Lang.AppError:
        MESSAGE twoError:GetMessage(1) VIEW-AS ALERT-BOX.
    END CATCH.
END. /* FIRST DO */

DO ON ERROR UNDO, LEAVE:
    FIND FIRST Customer WHERE CustNum = 6000.

    /* You can reuse an object-variable from a different
       associated block as long as it's the same type. */
    CATCH oneError AS Progress.Lang.SysError:
        MESSAGE oneError:GetMessage(1) VIEW-AS ALERT-BOX.
    END CATCH.

    /* NOT LEGAL: You cannot reuse an object variable name
       if its type is different than the variable's previous
       use within the same scope. */
    CATCH oneError AS Progress.Lang.AppError:
        MESSAGE oneError:GetMessage(1) VIEW-AS ALERT-BOX.
    END CATCH.
END. /* SECOND DO */

PROCEDURE foo:
    FIND FIRST Customer WHERE CustNum = 7000.

   /* This IS LEGAL because a new oneError variable will be
      defined within the scope of this subprocedure so its
      type does not have to match. */
    CATCH oneError AS Progress.Lang.AppError:
        MESSAGE oneError:GetMessage(1) VIEW-AS ALERT-BOX.
    END CATCH.
END.
```

Example 3

The following example illustrates using multiple CATCH blocks to handle specific conditions.The CATCH block for the more specialized error classes should come first.

```
FOR EACH Customer:

    < Code body of the associated block >

    /* This CATCH specifies the most specialized user-defined error class.
       It will catch only myAppError error objects or objects derived from
       myAppError. */

    CATCH eMyAppError AS Acme.Error.myAppError:
        /*Handler code for Acme.Error.myAppError condition. */
    END CATCH.

    /* This CATCH will handle Progress.Lang.AppError or any user-defined
       application error type, except for eMyAppError which is handled
       by the preceding CATCH block. */

    CATCH eAppError AS Progress.Lang.AppError:
        /* Handler code for AppError condition. */
    END CATCH.

    /* This CATCH will handle any error raised by an ABL statement.
       Since it is not in the class hierarchy of AppError, this CATCH
       could come before or after the CATCH for AppError */

    CATCH eSysError AS Progress.Lang.SysError:
        /* Handler code for SysError condition. */
    END CATCH.

    /* This is compatible with any condition object that
       implements the Progress.Lang.Error interface. All the
       above classes qualify, as well as a StopError object which
       is a SysError.  So, in this context, this CATCH block will
       only run for a .NET Exception. */

    CATCH eError AS Progress.Lang.Error:
        /* Handler code for any error condition. */
    END CATCH.

END. /* Associated Block */
```

Notes

- The use of CATCH blocks is supported for all blocks, however it cannot be used in a simple DO block (one with no options). This is because the simple DO block does not have implicit error handling capabilities. You will get a compiler error if you try. DO blocks must have an explicit TRANSACTION, ON ERROR UNDO, or ON STOP directive, in order to have a CATCH block.
- If a FINALLY block is used at the end of an associated block, it must come after any CATCH blocks. For more information on FINALLY blocks see *ABL Error Handling*.
- The compiler issues a warning message if a block contains a CATCH block that is not reachable. For example, if you have a CATCH block for Progress.Lang.AppError followed by a CATCH block for a subtype of Progress.Lang.AppError, you get a compiler warning.
- Using NO-ERROR on a statement prevents a CATCH block from executing if the statement raises a condition. The general error handling precedence, from highest to lowest, is the following:
  - Statement NO-ERROR option
  - CATCH block
  - Block's ON phrase (explicit or implicit)

For more information on the precedence of CATCH blocks see *ABL Error Handling*.

- The CATCH block is an undoable block with implicit ON ERROR UNDO, THROW error handling. You cannot explicitly override the ON ERROR directive for a CATCH block.
- A CATCH block within a CATCH block only handles errors raised within the CATCH block. To prevent infinite looping, the implicit or explicit UNDO, THROW statement within the top-level CATCH block, or any CATCH block nested within it, immediately throws the error to the block that encloses the associated block of the top-level CATCH block. For more detail, see *ABL Error Handling*.
- The ability to throw and catch Progress.Lang.Stop (and it's subclasses) or Progress.Lang.StopError objects is the default in 12.0. To turn this off you must have the Catch STOP (-catchStop) startup parameter set to the value 0 (-catchStop 0). If a value of 0 is specified, the AVM raises the STOP condition only, without throwing the associated object.
- Class-based error or stop objects can be thrown from an OpenEdge application server and handled by a CATCH block on an ABL client. To be thrown, any user-defined error classes must be defined, on both the server and client sides, as SERIALIZABLE. For more information on throwing error objects from an application server to an ABL client, see *ABL Error Handling*.

See also

BLOCK-LEVEL ON ERROR UNDO, THROW statement, ON ERROR phrase, ON STOP phrase, Progress.Lang.Error interface, Progress.Lang.Stop class, RETURN statement, ROUTINE-LEVEL ON ERROR UNDO, THROW statement, UNDO statement

CHOOSE statement

After you display data, the CHOOSE statement moves a highlight bar among a series of choices and selects a choice when you press **GO**, **RETURN**, or enter a unique combination of initial characters.

This statement is supported only for backward compatibility.

Note: Does not apply to SpeedScript programming.

Syntax

```
CHOOSE
  {    { ROW field  [ HELP char-constant ] }
    |  { FIELD { field [ HELP char-constant ] } ...}
  }
  [ AUTO-RETURN ]  [ COLOR color-phrase  ]
  [ GO-ON ( key-label  ... ) ]  [ KEYS char-variable ]
  [ NO-ERROR ]  [ PAUSE expression ]
  { [ frame-phrase ] }
```

You can specify the AUTO-RETURN, COLOR, GO-ON, KEYS, NO-ERROR, and PAUSE options in any order.

ROW *field*

Tells CHOOSE to move a highlight bar among iterations of a down frame. The *field* is the name of the field that you want the highlight bar to begin highlighting. The ROW option is useful for browsing through a set of records, although *field* does not have to refer to database records.

If you use the ROW option with the CHOOSE statement, use the SCROLL statement as well. See the SCROLL statement reference entry examples.

If you use ROW, you can add a COLOR statement to control the video display highlighting.

FIELD *field*

Tells CHOOSE to move a highlight bar through a set of fields or set of array elements in a frame. The *field* argument is the table record or array variable with fields or elements through which you want to move the highlight bar. These fields or array elements must be defined as ABL default FILL-IN widgets (not specified with the FILL-IN NATIVE option). The FIELD option is useful for building menus. You can also supply help for *field*.

HELP *char-constant*

Lets you provide help text for each field in a CHOOSE FIELD statement or for the entire CHOOSE ROW statement. For the CHOOSE ROW statement, the help text is displayed throughout the CHOOSE operation. For the CHOOSE FIELD statement, the help text you specify for a field is displayed whenever you move to the field.

AUTO-RETURN

Tells the AVM to use the selection when you enter a unique string of initial characters. When you use AUTO-RETURN and the user enters a unique string of initial characters, the AVM sets the value of LASTKEY to KEYCODE (return).

COLOR *color-phrase*

Specifies a video attribute or color for the highlight bar. Following is the syntax for *color-phrase*:

```
{    NORMAL
   | INPUT
   | MESSAGES
   | protermcap-attribute
   | dos-hex-attribute
   | { [ BLINK- ] [ BRIGHT- ]
       [fgnd-color] [ bgnd-color ] }
   | { [ BLINK- ] [ RVV- ] [ UNDERLINE- ] [ BRIGHT- ]
       [ fgnd-color ] }
   | VALUE ( expression )
}
```

For more information on *color-phrase*, see the COLOR phrase reference entry.

`GO-ON ( key-label ) . . .`

Names *key-labels* for keys that cause CHOOSE to return control to the procedure. If you do not use the GO-ON option, CHOOSE returns control to the procedure when the user presses **GO**, **RETURN**, **END-ERROR**, or types a unique substring when AUTO-RETURN is in effect. If you don't specify **F1**, **RETURN**, or **F4**, those keys are still GO-ON keys by default.

`KEYS char-variable`

If you want to highlight a particular choice when entering a CHOOSE statement, or if you want to know what keys the user pressed to make a selection, use the KEYS option. When you use the KEYS option, you must give the name of a character variable, *char-variable*. If *char-variable* is initialized to one of the choices before entering the CHOOSE statement, the AVM highlights that choice. As the user presses keys to move the highlight bar, the AVM saves those keystrokes in *char-variable*. You can test the value of *char-variable* after the CHOOSE statement returns control to the procedure. There is a 40-character limit when using the KEYS option.

`NO-ERROR`

Overrides default error handling by the CHOOSE statement, and returns control to the procedure. If you do not use the NO-ERROR option, the CHOOSE statement causes the terminal to beep when the user presses an invalid key.

If you use the NO-ERROR option and the user presses an invalid key, the CHOOSE statement ends. At this point, you usually want to use the LASTKEY function to test the value of the last key the user pressed and then take the appropriate action.

Note that the NO-ERROR option of the CHOOSE statement does not have any affect on the ERROR-STATUS system handle.

`PAUSE expression`

Specifies a time-out period in seconds. If the user does not make a keystroke for the specified number of seconds, the CHOOSE statement times out and returns control to the procedure. You can choose a whole or a fractional value for the time-out interval. If the time-out period you specify is a fractional value, the value is rounded to the nearest whole millisecond.

The time-out period begins before the user's first keystroke and is reset after each keystroke. If CHOOSE times out, the value of LASTKEY is -1. Use the time-out period to prevent inactivity.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

Example

The following procedure displays a strip menu with four choices. The procedure defines two arrays; one holds the items for selection on the menu, the other holds the names of the programs associated with the menu selections. The CHOOSE statement allows the user to select an item from the strip menu. The AVM finds the number (within the array) associated with the item selected and the program associated with that number in the proglist array. The AVM runs the program, if it exists, and displays a message. It also allows the user to select another item if the program does not exist. (In your own application, you associate actions with items selected by the CHOOSE statement.)

r-chsmnu.p

```
DEFINE VARIABLE menu     AS CHARACTER NO-UNDO EXTENT 4 FORMAT "x(7)"
  INITIAL [ "Browse", "Create", "Update", "Exit" ].
DEFINE VARIABLE proglist AS CHARACTER NO-UNDO EXTENT 4
  INITIAL [ "brws.p", "cre.p", "upd.p", "exit.p"].

FORM "Use the sample strip menu to select an action."
  WITH FRAME instruc CENTERED ROW 10.

REPEAT:
  VIEW FRAME instruc.
  DISPLAY menu WITH NO-LABELS ROW 21 NO-BOX
    FRAME f-menu CENTERED.
  HIDE MESSAGE.
  CHOOSE FIELD menu GO-ON (F5) AUTO-RETURN
    WITH FRAME f-menu.
  IF SEARCH(proglist[FRAME-INDEX]) = ? THEN DO:
    MESSAGE "The program" proglist[FRAME-INDEX] "does not exist.".
    MESSAGE "Please make another choice.".
  END.
  ELSE RUN VALUE(proglist[FRAME-INDEX]).
END.
```

The GO-ON option sets the **GET** key to perform an action like **GO**. With the LASTKEY function, you could check for **F5** and take another action relevant to your application.

Notes

- If you do not specify help text in the CHOOSE statement, any help text you specify for the field in the Data Dictionary is displayed instead. If no help text is specified in either the CHOOSE statement or Data Dictionary, then the status default message is displayed throughout the CHOOSE statement.
- The CHOOSE statement takes different actions depending on the key you press and whether you use the NO-ERROR option, as shown in the following table.

Table 12: CHOOSE statement actions

Key	NO-ERROR	Action
Valid cursor motion[5]	N/A	Clear saved keys and move highlight bar.
Invalid cursor motion[6]	NO	Clear saved keys and beep terminal.
Invalid cursor motion[6]	YES	Clear saved keys and return control to procedure.
A non-unique string followed by an alphanumeric character that does not form a matchable string[7]	NO	Clear saved keys and try to match the last key entered. If no match is available then beep terminal.

[5] Valid cursor motion keys within a frame are CURSOR UP, CURSOR DOWN, CURSOR RIGHT, CURSOR LEFT, SPACEBAR, TAB, and BACKTAB.

[6] Invalid cursor motion keys are CURSOR UP, CURSOR DOWN, CURSOR RIGHT, and CURSOR LEFT that cause the cursor to move outside the frame.

[7] The `r-chs1.p` procedure below, shows what the CHOOSE statement does when the user enters a non-unique string followed by a character that, together with the rest of the string, does not match anything.

Key	NO-ERROR	Action
A non-unique string followed by an alphanumeric character that does not form a matchable string with the other characters	YES	Return control to procedure.
An invalid string	NO	Beep terminal.
An invalid string	YES	Return control to the procedure and, if the KEYS option was used, save any printable keys.
Other keys[8]	NO	Beep terminal.
Other keys[8]	YES	Return control to procedure.

r-chs1.p

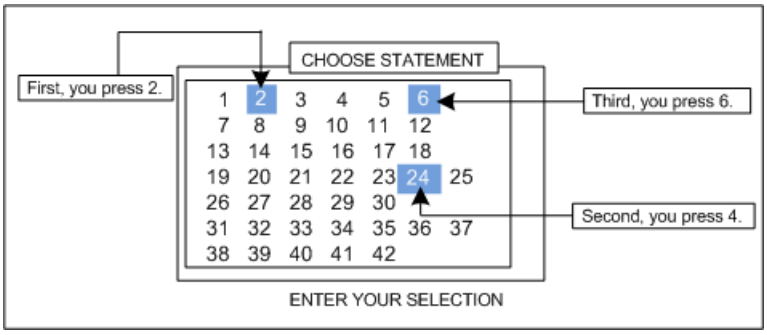
```
DEFINE VARIABLE abc AS CHARACTER NO-UNDO FORMAT "x(3)" EXTENT 42.
DEFINE VARIABLE ix  AS INTEGER   NO-UNDO.

DO ix = 1 TO 42:
  abc[ix] = STRING(ix, ">9").
END.

DISPLAY abc NO-LABELS WITH CENTERED ROW 4
  TITLE " CHOOSE STATEMENT " FRAME f-choose WIDTH 36.
DISPLAY "Enter your selection " WITH CENTERED NO-BOX
  FRAME f-instruct.
PAUSE 1 BEFORE-HIDE NO-MESSAGE.

REPEAT:
  HIDE MESSAGE.
  CHOOSE FIELD abc AUTO-RETURN WITH FRAME f-choose.
  MESSAGE "You selected -> " FRAME-VALUE.
END.
```

Once you run this procedure, your window looks like the following:



- When you press 2, CHOOSE moves the highlight bar to 2. When you press 4, CHOOSE moves the bar to 24. When you press 6, CHOOSE looks for the string 246. Because it cannot find the string, it matches the last key pressed (6) and places the highlight bar on 6.

[8] Other keys are non-cursor-motion, non-alphanumeric keys (function keys, BACKSPACE) except for: HELP, STOP, RETURN, GO, END, ERROR, END-ERROR. Keys defined to do the actions of these keys still do so.

- A choose field can temporarily become a handle type for internal purposes, but is not actually a widget since it does not have its own set of attributes and widgets. Therefore, you might see myhandle:TYPE = choose field in the widget tree, but you cannot manipulate the choose field.
- In the context of the .NET blocking method, `System.Windows.Forms.Application:Run( )`, if you execute the CHOOSE statement for a non-modal ABL window from within a user-defined function or non-VOID method, this statement raises the STOP condition. For more information on the .NET `Application:Run( )` method, see the reference entry for the WAIT-FOR statement (.NET and ABL).

See also

COLOR phrase, Frame phrase, SCROLL statement, STATUS statement

CHR function

Converts an integer value to its corresponding character value.

Syntax

```
CHR ( expression
      [ ,target-codepage [ , source-codepage ] ]
    )
```

expression

An expression that yields an integer value that you want to convert to a character value.

If the value of *expression* is in the range of 1 to 255, CHR returns a single character. This character might not be printable or might not display on certain terminals. For a value greater than 255, the CHR function checks whether the value represents a valid character within the source code page. For a double-byte or Unicode source code page, it checks for a corresponding lead-byte value. If the integer value corresponds to a valid lead-byte and the value represents a valid character in the source code page, CHR returns the character. Otherwise, the character returns a null string.

target-codepage

A character-string expression that evaluates to the name of a code page. The name that you specify must be a valid code page name available in the `DLC/convmap.cp` file (a binary file that contains all of the tables that ABL uses for character management). If you supply a non-valid name, the CHR function returns a null string. Before returning a character value, the CHR function converts *expression* from *source-codepage* to *target-codepage*. The returned character value is relative to *target-codepage*. If you do not specify *target-codepage*, no code page conversions occur.

source-codepage

A character-string expression that evaluates to the name of a code page. The name that you specify must be a valid code page name available in the `DLC/convmap.cp` file. If you supply a non-valid name, the CHR function returns a null string. The *source-codepage* specifies the name of the code page to which *expression* is relative. The default value of *source-codepage* is the value of SESSION:CHARSET.

Example

The `r-chr.p` procedure initializes the 26 elements of the letter array to the letters A through Z.

r-chr.p

```
DEFINE VARIABLE ix     AS INTEGER   NO-UNDO.
DEFINE VARIABLE letter AS CHARACTER NO-UNDO FORMAT "X(1)" EXTENT 26.

DO ix = 1 TO 26:
  letter[ix] = CHR((ASC("A")) - 1 + ix).
END.

DISPLAY SKIP(1) letter WITH 2 COLUMNS NO-LABELS
  TITLE "T H E  A L P H A B E T".
```

Notes

- The CHR function returns the corresponding character in the specified code page. By default, the value of SESSION:CHARSET is iso8859-1. You can set a different internal code page by specifying the Internal Code Page (-cpinternal) parameter. For more information, see *Startup Command and Parameter Reference*.
- The CHR function is double-byte enabled. For a value greater than 255 and less than 65535, it checks for a lead-byte value. If the lead-byte value is valid, the AVM creates and returns a double-byte character.

See also

ASC function, CODEPAGE-CONVERT function, SESSION system handle, STRING function

Class-based data member access

References a data member of a class in order to read or write its value.

Syntax

```
[ [ class-type-name | object-reference ] : ] data-member-name
  [[ index ]]
```

class-type-name

The name of an ABL or .NET class type that defines the specified data member as a static member. The use of *class-type-name* to access a static data member is optional when you access the data member from within the class hierarchy where it is defined. For more information, see the notes for this reference entry. You can use *class-type-name* only to access data members that can be defined as PUBLIC. For more information, see the notes for this reference entry. You cannot use *class-type-name* to access an instance data member. For more information on specifying class (object) type names, see the Type-name syntax reference entry. You also can use the unqualified class name with the presence of an appropriate USING statement.

object-reference

Specifies a reference to an instance of an ABL or .NET class (an object) that defines the specified data member as an instance member. The use of *object-reference* to access an instance data member is optional when you access the data member from within the class hierarchy where it is defined. For more information, see the notes for this reference entry.For more information, see the notes for this reference entry. You cannot use *object-reference* to access a static data member. For information on specifying object references, see the reference entry for a Class-based object reference.

data-member-name

The name of a data member you want to access. In .NET, a data member is referred to as a field. A data member is a data element that is defined in, and at, the level of a class definition (as distinct from a local data element that is defined in and at the level of a method definition). If it is an instance data member, a separate copy exists for each instance of its defining class, for as long as the class instance exists. If it is a static data member, only one copy exists for the defining class type for the entire ABL session, regardless if any instance of the class exists. A data member is available either inside or outside of the class hierarchy depending on its access mode.

[*index*]

Specifies the index to reference an element of an ABL array data member, where *index* is an integer value that identifies the element. The brackets are a required part of the index syntax.

Note: To access the elements of a .NET array object, you must use the .NET `SetValue( )` and `GetValue( )` methods on the array object instead. For more information on using these methods, see the Microsoft .NET Framework documentation and Parameter passing syntax.

Examples

The following code fragment shows a reference to a public instance data member (`hHighCustData`) on an instance of the sample class, `r-CustObj`:

```
DEFINE VARIABLE rObj AS CLASS r-CustObj NO-UNDO.
rObj = NEW r-CustObj( ) NO-ERROR.

MESSAGE "Number of buffers =" rObj:hHighCustData:NUM-BUFFERS
  VIEW-AS ALERT-BOX.
```

This instance data member is a handle to a ProDataSet that is used, in this case, to reference the NUM-BUFFERS attribute, which returns the number of buffers in the ProDataSet.

The following code fragment shows a reference to a public static data member (`hHighCustData`) on the sample class type, `r-CustObjStatic`:

```
MESSAGE "Number of buffers =" r-CustObjStatic:hHighCustData:NUM-BUFFERS
  VIEW-AS ALERT-BOX.
```

This static data member is also a handle to a similar ProDataSet, and example code performs the same function as the previous instance code without having to instantiate a class.

For more information on these data members and the sample classes in which they are defined, see the examples in the CLASS statement reference entry.

Notes

- Using the appropriate syntax, you can read and write data as defined for the data member type. For more information on the supported ABL data member types, see the reference entry for the CLASS statement and the DEFINE statement for each type of data member.
- You can also read or write a .NET data member, depending on its definition. For example, if a given .NET data member is defined as a constant, you can only read the defined value of that data member, but you cannot write to it. .NET data members can also be defined with a variable value as read-only. For information on the definition of a .NET data member, see its field entry in Microsoft .NET, Infragistics, or another third-party class library that defines it. For information on how .NET data types map to ABL data types, see the Data types reference entry.
- If the data member is defined as static, you can access the data member using static type-name syntax whether or not an instance of its defining class exists.
- ABL implicitly defines the default buffers of any database tables you reference within a class definition as private instance data members of the class.
- If you reference an available data member within a static constructor, static method, static property accessor, or as part of another static data member definition that is defined in the same class or class hierarchy as the referenced member (such as referencing a buffer in a static data-source definition), the referenced data member must also be defined as static; attempting to directly reference an instance data member that is defined in the same class or class hierarchy as the referencing static member raises a compile-time error. To reference a buffer for a database table in a static data member definition, you must define and reference an alternate static buffer data member for the table with a compatible access mode.
- If a data member definition includes a reference to another data member defined in the same class, the access mode of the referenced data member must be no less restrictive than the access mode of the data member definition where it is referenced. For example, if a data-source is defined with a PROTECTED access mode, any buffer that it references must be defined as either PROTECTED or PUBLIC.
- From within an ABL class definition, you can reference any data member that is both defined and available within the class hierarchy using its *data-member-name* without a qualifying *object-reference* (for instance, data members) or *class-type-name* (for static data members). Data members that you can reference directly by *data-member-name* within the class hierarchy where they are defined include:
  - Class-based variables (and their associated visual widgets). You can use the THIS-OBJECT system reference as an *object-reference* to access any available instance variable data member that is defined within the class hierarchy. However, you **must** use THIS-OBJECT to access any instance variable data member whose name is a reserved keyword. You can also use *class-type-name* to access any available static variable data member that is defined within the class hierarchy. However, you **must** use *class-type-name* to access any static variable data member whose name is a reserved keyword.
- Streams, work-tables, and widgets, including browses, buttons, frames, images, menus, sub-menus, and rectangles that you define in a class block are not data members. Like triggers, these are class-scoped resources that are privately available for use with both instance and static members within the defining class.
- From within a class hierarchy where static data members are defined, you can use a qualifying *class-type-name* to access only the static variable data members from both ABL and .NET, including the elements of arrays. To access all other static data members (such as temp-tables or buffers) defined in the current class hierarchy, you must only use an unqualified *data-member-name*.
- From outside a class hierarchy, you must reference any available instance data member using its *data-member-name* qualified by an *object-reference* to the class instance where the data member is defined, and you must reference any available static data member using the *class-type-name* of the class where the data member is defined as static. The only data members that are available outside the class hierarchy where they are defined are variables defined as PUBLIC (and any associated visual widgets for instance variables), including the elements of arrays.

Note: From outside the class hierarchy where they are defined, you cannot apply events (using the APPLY statement) to triggers defined for widgets associated with PUBLIC data members.

See also

Class-based object reference, CLASS statement, Type-name syntax

Class-based method call

Invokes a method of a class. If the method returns a value, the method call can appear anywhere that an expression can appear, and it can also appear as a single statement, ignoring the return value. If the method is VOID (does not return a value), the method call must appear as a single statement.

Syntax

```
[ { class-type-name | object-reference } : ]
  method-name ( [ parameter [ , parameter ] ... ] ) [ NO-ERROR ]
```

class-type-name

The name of an ABL or .NET class type that defines the specified method as a static member. The use of *class-type-name* to call a static method is optional when you call the method from within the class hierarchy where it is defined. For more information, see the notes for this reference entry. You cannot use *class-type-name* to call an instance method. For more information on specifying class (object) type names, see the Type-name syntax reference entry. You can use the unqualified class name with the presence of an appropriate USING statement.

object-reference

Specifies a reference to an ABL or .NET class instance (an object) that defines the specified method as an instance member. The use of *object-reference* to call an instance method is optional when you call the method from within the class hierarchy where it is defined. For more information, see the notes for this reference entry. You cannot use *object-reference* to call a static method. For information on specifying object references, see the reference entry for a Class-based object reference.

method-name

Specifies the name of an ABL or .NET class method you want to call. A class method is a named block of ABL or .NET code, similar to a procedure or user-defined function, that is defined in a class. An instance method is available for an instance of the class for as long as the class instance exists. A static method is available for the defining class type during the entire ABL session, regardless if an instance of the class exists. A class method is available inside or outside of the class hierarchy depending on its access mode.

```
( [ parameter [ , parameter ] ... ] )
```

Specifies zero or more parameters passed to the method. You must provide the parameters identified by the specified method, matched with respect to number, data type, and mode. To invoke a method

that is overloaded in the class, you must specify sufficient information for each *parameter* in order to disambiguate methods that have similar parameter lists. Otherwise, the AVM raises an error identifying the ambiguity.

For more information on parameter passing syntax and on disambiguating overloaded ABL and .NET methods, see the Parameter passing syntax reference entry.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Examples

The following code fragment shows a call to a public instance method (SetHighCustomerData( )) on an instance of the sample class, r-CustObj:

```
DEFINE VARIABLE rObj AS CLASS r-CustObj NO-UNDO.
rObj = NEW r-CustObj( ) NO-ERROR.

rObj:SetHighCustomerData( ) NO-ERROR.
```

This instance method initializes instance data for the class.

The following code fragment shows a call to a public static method (SetHighCustomerData( )) on the sample class type, r-CustObjStatic:

```
r-CustObjStatic:SetHighCustomerData( ) NO-ERROR.
```

This static method initializes class static data without having to instantiate the class, as in the previous instance code.

For more information on these methods and the sample classes in which they are defined, see the examples in the CLASS statement reference entry.

Notes

- Using the appropriate syntax, you can invoke the method as a statement (without returning a value) or invoke the method in an expression that uses the defined return value according to its data type. For information on ABL methods and their definition, see the METHOD statement reference entry. For information on the definition of a .NET method, see its entry in the Microsoft .NET, Infragistics, or other class library that defines it. For information on how .NET data types map to ABL data types, see the Data types reference entry.
- If the method is defined as static, you can call the method whether or not an instance of its defining class exists.
- If you reference an available method within a static constructor, static method, or static property accessor that is defined in the same class or class hierarchy as the referenced method, the referenced method must also be defined as static; attempting to directly reference an instance method that is defined in the same class or class hierarchy as the referencing static member raises a compile-time error.
- From within an ABL class definition, you can typically invoke any instance method that is defined and available within the class hierarchy by referencing its *method-name* and any parameter list without a qualifying *object-reference*. However, if the method name is a reserved keyword, you must call the method using THIS-OBJECT as the *object-reference*. If the instance method is defined as an OVERRIDE

method, you can invoke the method implementation that it overrides in the most derived class where it is defined using SUPER as the *object-reference*. For more information, see the reference entry for the SUPER system reference.

- From within a class definition, you can typically invoke any static method that is defined and available within the class hierarchy by referencing its *method-name* and any parameter list without a qualifying *class-type-name*. However, if the method name is a reserved keyword, you must call the method using the qualifying *class-type-name*, even if the method is called from within the class that defines it. If the static method is defined as an OVERRIDE (redefining) method, you can invoke any static method implementation in the class hierarchy that it redefines by using the qualifying *class-type-name* of the class that defines the particular method implementation you want to call.
- From outside a class hierarchy, you can only invoke an available instance method by referencing its *method-name* and its parameters qualified by an *object-reference* to the class instance where the method is defined; you can only invoke an available static method by referencing its *method-name* and its parameters qualified by the *class-type-name* of the class that defines the method as static.
- A method marked as PRIVATE can be accessed from any instance of the class that defines it. A PACKAGE-PRIVATE method can be accessed from any instance of the class that defines it or any class in the same package. A PROTECTED method can be accessed from any instance of the class that defines it or a subclass of the defining class. A PACKAGE-PROTECTED method can be accessed from any instance of the class that defines it, a subclass of the defining class (which may be in different packages), or any class in the same package. A PUBLIC method can be accessed from any class (whether in the same package or another package), or from a procedure file.
- You can call an abstract method either from within the class that defines it or on an *object-reference* defined as the type of the abstract class that defines it. Although an abstract method is defined without an implementation, at run time, the method is always implemented in a derived class. So, any reference to an abstract method is always resolved by the most derived class that implements it.
- ABL has the following limitations on the .NET methods you can call:
  - You cannot call the static method, `System.Windows.Forms.Application:DoEvent( )`. In ABL, you can use the PROCESS EVENTS statement to handle both ABL and .NET events.
  - You cannot call the static method, `System.Windows.Forms.Application:Run( )`, outside of a WAIT-FOR statement. For more information, see the WAIT-FOR statement (.NET and ABL) reference entry.
  - You cannot call the instance method, `System.Windows.Forms.Form:ShowDialog( )`, outside of a WAIT-FOR statement. For more information, see the WAIT-FOR statement (.NET and ABL) reference entry.
  - You cannot call a .NET generic method. Similar to a .NET generic type, a .NET generic method has a name that is appended with a comma-separated list of one or more type parameters enclosed in angle brackets, for example, `Add<T, S>(tVar, sVar)`. In this case, you would substitute .NET data types for the parameters `T` and `S` when you called the method. ABL does not currently support any reference to these methods. For more information on ABL support for .NET generic types, see the Data types reference entry.

See also

Class-based object reference, METHOD statement, NO-ERROR option, Parameter passing syntax, SUPER system reference, THIS-OBJECT system reference, Type-name syntax

Class-based object reference

References an instance of an ABL or .NET class and provides access to instance class members (per the restrictions of the access levels assigned to the members).

Syntax

```
object-reference [ [ key ] ] [ : { data-member
                                 | property
                                 | method
                                 }
                               ]
```

object-reference

Specifies a reference to an instance of a class (an object). This can be a variable (data member), property, data member or property array element, handle attribute (such as FIRST-OBJECT), method return value, ABL function return, or user-defined function return value defined for an ABL or .NET object type. It also can be an appropriately-used system reference, such as SUPER or THIS-OBJECT. Its value references an instance of a specified class (*class-based object*). If it is for a class, the data type of the value can be for the class type itself or for a super class of the referenced class. If it is for an interface, its value can reference an instance of any class that implements that interface.

[*key*]

Specifies an indexer for the default indexed property of a .NET class. The brackets are a required part of the syntax.

The *key* value is an index value of any supported .NET data type that identifies the property value you want to reference. If this value is an object reference, you can also use it to access any public instance *data-member*, *property*, or *method* on the .NET class instance that the property references. For more information, see the reference entry for a Class-based property access.

Note: ABL also allows you to access the default indexed property of a class by specifying the indexer on the default property name. There is no functional difference between specifying the indexer directly on the object reference and specifying it on the property name. For more information on accessing .NET indexed properties, see the Class-based property access reference entry.

---

data-member

Specifies the name of a PUBLIC instance variable defined in the specified ABL class or of a public instance field defined in the specified .NET class. The value of the ABL variable data member can be written or read for the class instance referenced by *object-reference*. The value of a .NET field can be read or written for the class instance specified by *object-reference*, unless it is defined as a constant (using the C# `const` keyword). If it is a constant, it can only be read. For more information, see the reference entry for a Class-based data member access.

property

Specifies the name of a PUBLIC instance property defined in the specified ABL or .NET class. The value of this property can be written or read for the class instance referenced by *object-reference*, depending on the property definition. For more information, see the reference entry for a Class-based property access.

method

Specifies the name and any run-time parameters taken by a PUBLIC instance method defined in the specified ABL or .NET class, or a call to a built-in event method. A class method can provide a value, depending on the method definition. For more information on calling class methods, see the reference entry for a Class-based method call. For more information on the built-in event methods, see the Class Events Reference.

Notes

- You can obtain an object reference for a .NET object in the same way you can obtain a reference to any ABL class-based object, by creating a new object or obtaining the reference to an existing one. You cannot create or reference instances of the following .NET classes:
  - Classes defined in the .NET default namespace (that is, classes with no namespace defined)
  - `System.Threading.Thread` or any class derived from it
  - `System.Delegate` or any delegate type derived from it
- If the documentation for a .NET class includes a member in a list of "Explicit Interface Implementations", you cannot access that member on an object reference defined as the class type. You can only access that member on an object reference that is defined as (or cast to) a particular interface that defines the method. Thus, ABL documentation refers to such a .NET class member as an *explicit interface member*.
- You cannot define or otherwise obtain an object reference to a .NET mapped object type (for example, `System.Int32`), except when overriding a .NET method or when implementing a method or property of a .NET interface. You can only access instances of .NET mapped object types as corresponding ABL primitive types (for example, INTEGER). For more information, see the Data types reference entry. However, you can access .NET static properties and methods on their defining .NET mapped object types.
- ABL classes do not currently support default or indexed properties. So, you cannot reference an ABL class instance using an indexer for a default indexed property.

See also

Class-based data member access, Class-based method call, Class-based property access, CLASS statement, FIRST-OBJECT attribute, LAST-OBJECT attribute, NEW function (classes), SUPER system reference, THIS-OBJECT system reference

Class-based property access

References a property of a class in order to read or write its value.

Syntax

```
[ [ class-type-name | object-reference ] : ]
  property-name [ [ index | key ] ]
```

class-type-name

The name of an ABL or .NET object type that defines the specified property as a static member. The use of *class-type-name* to access a static property is optional when you access the property from within the class hierarchy where it is defined. For more information, see the notes for this reference entry. You cannot use *class-type-name* to access an instance property. For more information on specifying object type names, see the Type-name syntax reference entry. You can use the unqualified class name with the presence of an appropriate USING statement.

object-reference

Specifies a reference to an instance of an ABL or .NET class (an object) that defines the specified property as an instance member. The use of *object-reference* to access an instance property is optional when you access the property from within the class hierarchy where it is defined. For more information, see the notes for this reference entry. You cannot use *object-reference* to access a static property. For information on specifying object references, see the reference entry for a Class-based object reference.

property-name

The name of an ABL or .NET property. Accessing a property is similar to accessing a variable data member. Also, like a data member, a property is defined in, and at, the level of a class definition. If it is an instance property, a separate copy exists for each instance of its defining class, for as long as the class instance exists. If it is a static property, only one copy exists for the defining class type during the entire ABL session, regardless if any instance of the class exists. A property is available inside or outside of the class hierarchy depending on its access mode. If and how a property can be read or written depends on a combination of its access mode and associated behavior defined for the property. For more information on ABL property definitions, see the DEFINE PROPERTY statement reference entry. For information on the definition of a .NET property, see its entry in Microsoft .NET, Infragistics, or another class library that defines it.

[*index* | *key*]

An *index* specifies an index into an ABL array property. A *key* specifies an indexer for a .NET indexed property. The brackets are a required part of the syntax.

If an array *index* is specified, it can be an integer value that identifies an element of an ABL array property, similar to an element of an ABL array data member (see the Class-based data member access reference entry).

If a *key* for a .NET property indexer is specified, it can be a value of any supported .NET data type that identifies the property value to reference. A .NET indexed property represents a group of properties, where the *key* is a value that indicates what member of the group is being referenced. A .NET indexed property can have more than one type of indexer that is overloaded according to the data type of its *key*, similar to a method that is overloaded by a single parameter. However, you cannot specify an AS data type with the *key* in order to identify a particular .NET indexer overload, as you can with a .NET method parameter (see the Parameter passing syntax reference entry). Also, some .NET indexed properties have indexers with multiple *key* values, each with its own data type. However, ABL only supports .NET indexed properties that have single-*key* indexers. So, you might be able to use some indexers for a property and not be able to use others.

Thus, ABL only recognizes a .NET property indexer if the property has a single-*key* indexer. If so, ABL selects the indexer using the ABL data type that you specify for the *key*, in order by the following criteria:

1. If the .NET data type of an indexer *key* is the default match for the specified ABL data type, ABL uses that *key*.
2. If there is no .NET indexer *key* that is a default match for the specified ABL data type, ABL uses the first *key* that it encounters where the .NET data type is an implicit data type mapping for the specified ABL data type.

For more information on how ABL data types map to .NET data types, see the Data types reference entry.

For example, if you specify an ABL INTEGER value for the *key*, and the indexer is overloaded by .NET `System.Double`, `System.Byte`, and `System.Int32` keys, ABL uses the `System.Int32`*key* to index the property. If the available overloadings are `System.Double`, `System.Byte`, and `System.Int16`, ABL uses either the `System.Byte` or the `System.Int16`*key*, which ever is encountered first.

ABL classes do not currently support the definition of indexed properties. So, you cannot access ABL properties using an indexer.

Examples

The following code fragment shows a reference to a public instance property (`HighCustBalance`) on an instance of the sample class, `r-CustObj`:

```
DEFINE VARIABLE rObj AS CLASS r-CustObj NO-UNDO.
rObj = NEW r-CustObj( ) NO-ERROR.

MESSAGE "High customer balance =" rObj:HighCustBalance
  VIEW-AS ALERT-BOX.
```

Once initialized, this instance property returns the highest balance for all `Customer` records in the `Sports2020` database.

The following code fragment shows a reference to a public static property (`HighCustBalance`) on the sample class type, `r-CustObjStatic`:

```
MESSAGE "High customer balance =" r-CustObjStatic:HighCustBalance
  VIEW-AS ALERT-BOX.
```

This static property similarly returns the highest balance for all `Customer` records in the `Sports2020` database without having to instantiate the class, as in the previous instance code.

For more information on these properties and the sample classes in which they are defined, see the examples in the CLASS statement reference entry.

Notes

- Using the appropriate property syntax, if the property is defined as both readable and writable, you can assign and otherwise read and write the value of a property in the same way as a variable data member of the same data type. If the property is read-only, you can only read the property value; if the property is write-only, you can only write a value to the property. For information on how .NET data types map to ABL data types when using .NET properties, see the Data types reference entry.
- When and how the value of a property can be read or written depends on the definition of special methods (accessors) that execute during property access. For more information on accessors for ABL properties, see the DEFINE PROPERTY statement reference entry. Note that even though the value written to a property appears to its SET accessor like a method parameter, the compatibility between the data type of the written value and the data type of the property follows the same rules as the assignment of data types for variables or data members. In other words, ABL attempts to identify an appropriate conversion when you assign a value to a property. For more information, see the Assignment (=) statement reference entry.

  For more information on accessors for a .NET property, see its entry in Microsoft .NET, Infragistics, or another class library that defines the property. In C# documentation, the presence of a get accessor in the property signature means that the property is readable; the presence of a set accessor in the property signature means that the property is writable. For example, the signature of the .NET `HelpButton` property indicates that it is both readable and writable, because it shows both a get and a set accessor:

```
public bool HelpButton { get; set; }
```

- If the property is defined as static, you can access the property whether or not an instance of its defining class exists. This is done by qualifying the property name with the *`class-type-name`*.
- If you reference an available property within a static constructor, static method, or static property accessor that is defined in the same class or class hierarchy as the referenced property, the referenced property must also be defined as static; attempting to reference an instance property that is defined in the same class or class hierarchy as a referencing static member raises a compile-time error.
- In ABL, access modes are enforced at the class level. This means that if a class type has access to a class member, that member can be referenced via any instance of the class when done so from within the class implementation.
- From within an ABL class definition, you can reference any property that is both defined and available within the class hierarchy using its *`property-name`* without a qualifying *`object-reference`* or *`class-type-name`* (for static properties). Optionally, you can use the THIS-OBJECT system reference as an *`object-reference`* to access any available instance property defined within the class hierarchy. If the property name is a reserved keyword, you must reference the property by qualifying the name with either its *`class-type-name`* or THIS-OBJECT as the *`object-reference`*. For more information, see the reference entry for the THIS-OBJECT system reference.
- From outside a class hierarchy, you must reference any available property using its *`property-name`* qualified by an *`object-reference`*.
- You can access an abstract property either from within the class that defines it or on an *`object-reference`*. defined as the type of the abstract class that defines it. Although an abstract property is defined without an implementation, at run time, the property is always implemented in a derived class. So, any reference to an abstract property is always resolved by the derived class that implements it.
- When you access an ABL property from within its own definition (from an accessor), this access either directly accesses the property's default memory or invokes an accessor, depending on whether you are

reading or writing the property and which accessor you are accessing the property from. For more information, see the DEFINE PROPERTY statement reference entry.

- .NET indexed properties can be overloaded only by the indexer, and ABL identifies the overloading to use only by the definition of its indexer. Each overloading of an indexed property can specify a different access mode (for example, private or public) and can have different accessor method definitions. For example, one overloading of a property might be read-only and another overloading of the property might be write-only. In general, all overloadings of an indexed property share the same property data type. However, the data type of the index *key* does not have to match the data type of the property.
- A .NET class can have a default indexed property. By convention, most default indexed properties in the .NET Framework are named `Item`. Like some .NET languages, ABL allows you to access the default indexed property by its name, like any other property. However, ABL also allows you to access the default indexed property of a class by specifying the indexer directly on *`object-reference`*, omitting the property name (see the Class-based object reference reference entry in this book). For example, the .NET `System.Data.DataView` class defines a default `Item` property that returns a .NET `DataRow` given a record index. In ABL, therefore, the following indexed object references to a `DataView` class instance are identical:

```
DEFINE VARIABLE rDataView AS CLASS System.Data.DataView.
DEFINE VARIABLE rRow      AS CLASS System.Data.DataRow.
. . .
rRow = rDataView[2].
rRow = rDataView:Item[2].
```

  In .NET, the most common use of a default indexed property is to access an instance of a class from a collection. Two examples are the .NET `System.Data.DataTableCollection` and the `System.Data.DataRow` classes. For more information on working with .NET collections in ABL, see *Use GUI for .NET Objects in ABL Applications*.

- When you access a property, the property accessor can raise the ERROR condition on the statement that accesses the property. If this statement is defined with the NO-ERROR option, you can locate the property error information using the ERROR-STATUS system handle following statement execution, and you can read any returned error string using the RETURN-VALUE function. Alternatively, you can catch the property error object using a CATCH statement in the block that executes the statement. To access more comprehensive error information for a .NET exception, use a CATCH statement instead of the NO-ERROR option to handle .NET exceptions raised from accessing .NET properties. For more information on handling .NET exceptions, see the sections on .NET error handling in *Use GUI for .NET Objects in ABL Applications*.

See also

Class-based data member access, Class-based object reference, DEFINE PROPERTY statement, Type-name syntax

CLASS statement

Defines a user-defined class. A class defined with this statement represents a user-defined data type whose characteristics are defined by class members, including a set of class data members and properties that define class data, and a set of class methods and events that define class behavior. In addition to class members, a class definition can include special optional methods (one or more constructors and one destructor). It can also include class-scoped handle-based objects that provide private resources to the class, function prototypes for user-defined functions referenced by the class, and triggers to handle events for widgets and other handle-based objects defined by the class.

You can instantiate a non-abstract class as an object using the NEW function (classes), NEW statement, the New( ) method of the Progress.Lang.Class class, or the DYNAMIC-NEW statement. Instantiating a class returns an object reference that provides access to instance members of the class. You can also access static members of a class without instantiating the class.

Note: This statement is applicable only when used in a class definition (`.cls`) file. For more information, see the Notes section in this reference entry.

Syntax

```
CLASS class-type-name [ INHERITS super-type-name]
  [ IMPLEMENTS interface-type-name [ , interface-type-name ] ... ]
  [ USE-WIDGET-POOL ]
  [ ABSTRACT | FINAL ]
  [ SERIALIZABLE ]:
  class-body
```

class-type-name

Defines the type name for a user-defined class type. Specify an ABL class type name as described in the Type-name syntax reference entry. You can also specify an unqualified class name (without a package), but only if the class name represents the full type name for the class (that is, the class is not defined as part of a package).

Note that you cannot specify `Progress` as the first component of the package name for any ABL user-defined class. For example, `Progress.Inventory.UpdateInv` is an invalid type name for a user-defined class and results in a compiler error.

The value of *class-type-name* is restricted to alphanumeric characters plus the symbols #, $, %, and _.

INHERITS *super-type-name*

Optionally specifies the type name of a super class whose state and behavioral characteristics this user-defined class inherits. The new class becomes a subclass (derived class) of the specified super class and of every other super class in its class hierarchy. The *super-type-name* you specify can be an ABL or .NET class type name as described in the Type-name syntax reference entry. With an appropriate USING statement, you can also specify an unqualified class name (a class name without the ABL package or .NET namespace that completes the class type name).

If *super-type-name* is an abstract class, in this class definition you must implement all the abstract members (properties, methods, and events) that you inherit unless you also define this class as abstract.

The following files must be found at compile time:

- For an ABL super class, the class definition (`.cls`) file.
- For a .NET super class, the assembly that contains the .NET class definition. OpenEdge places restrictions on the .NET classes you can inherit. For more information on the compilation requirements for inheriting .NET classes, see the Notes of this reference entry.

By default, a class definition inherits directly from the built-in ABL root class, `Progress.Lang.Object`. If you inherit from a .NET class, its .NET root class, `System.Object`, also inherits directly from `Progress.Lang.Object`.

`IMPLEMENTS` *`interface-type-name`* **[** *`, interface-type-name`* **]...**

Optionally specifies the type name of one or more interfaces the new class implements. The new class must implement all properties and methods defined in the specified interfaces. Specify *`interface-type-name`* as an ABL or .NET interface type name as described in the Type-name syntax reference entry. With an appropriate USING statement, you can also specify an unqualified interface name (an interface name without the ABL package or .NET namespace that completes the interface type name). The value of *`interface-type-name`* is restricted to alphanumeric characters plus the symbols #, $, %, and _.

For each interface that you implement, your class must implement all the properties, methods, and class events defined by the interface. If any super class in the hierarchy implements a property, method, or event defined by a specified *`interface-type-name`*, this satisfies the requirement to implement that property, method, or event in your class.

If multiple implemented interfaces define an identical member, they all share the same implementation of that member. Thus, ABL classes do not support .NET explicit interface members.

ABL must find the following files at compile time:

- For an ABL interface, the class definition (`.cls`) file that contains the interface definition. For more information on defining an ABL interface, see the INTERFACE statement reference entry.
- For a .NET interface, the assembly that contains the .NET interface definition. OpenEdge places restrictions on the .NET interfaces you can implement. For more information on the compilation requirements for implementing .NET interfaces, see the Notes of this reference entry.

`USE-WIDGET-POOL`

Directs the AVM to create an unnamed widget pool that is scoped to the class hierarchy of each instance of the class. When specified, all dynamic handle-based objects that are created in the class by and for instance class members are created by default in this unnamed widget pool. If the class contains static members, this option directs the AVM to create a separate unnamed widget pool that is scoped to the class for all dynamic handle-based objects created for or by these static class members in the session.

If methods or externally invoked procedures create their own unnamed widget pools, dynamic handle-based objects are created in the most locally scoped unnamed widget pool, regardless of this option. If no unnamed widget pools are created, either for the class (no USE-WIDGET-POOL option specified) or within any invoked procedure or method, all dynamic objects are created by default in the session unnamed widget pool both for instance and static members.

Note: When you define an unnamed widget pool in a method or external procedure, it is scoped to that method or procedure. You can delete the widget pool within the method or procedure using the DELETE WIDGET-POOL statement, or let the AVM delete the widget pool when the method or procedure ends.

Within an instantiated class, any class in the class hierarchy that specifies the USE-WIDGET-POOL directs the AVM to create one unnamed widget pool that all classes in the class hierarchy share. The AVM deletes this unnamed widget pool when the class instance is deleted (destroyed).

You can also define a named widget pool in a class definition file. If you define a named widget pool, object instances are created in that widget pool only if you explicitly reference the widget pool by name.

For more information about defining and using widget-pools, see the CREATE WIDGET-POOL statement reference entry, and for creating widget pools within a class, see *Develop Object-oriented ABL Applications*.

ABSTRACT

Defines the class as abstract. An abstract class allows you to define any number of instance property, method, or event members, as abstract. Abstract members are prototypes, similar to interface members, without an implementation. Abstract members must be implemented by a derived class to provide data or behavior, but they also function polymorphically in the class hierarchy wherever they are still defined as abstract. So, for example, an abstract property can be accessed or an abstract method can be called and the result depends on its implementation further down in the class hierarchy. Also, whether or not an abstract class defines abstract members, it must be inherited by a derived class. Thus, you cannot directly instantiate an abstract class or define an abstract class as FINAL.

An abstract class can inherit from another abstract class or a non-abstract class, and it can redefine any implemented methods that it inherits as abstract. However, it cannot redefine any implemented properties or events that it inherits as abstract. Any redefined abstract methods must, again, be implemented in a derived class.

If a non-abstract class inherits from an abstract class, it must implement all of the inherited abstract members that are not already implemented in the abstract class hierarchy. An abstract class can contain implemented class members, either exclusively or in addition to abstract members. If an immediate subclass of an abstract class does not implement all of its inherited abstract members, the immediate subclass must also be defined as abstract, and a further derived class must implement the remaining abstract members.

FINAL

Indicates the class cannot be inherited by another class. That is, it cannot be used in the INHERITS phrase in another class definition. Thus, a FINAL class cannot also be defined as ABSTRACT. Define a class as FINAL when you do not want any of its state or behavior overridden.

SERIALIZABLE

Indicates that object instances of the class can be:

- serialized to a binary or JSON stream
- passed between an application server and a remote ABL client.

For a class to be serializable, all classes in the hierarchy must be marked as SERIALIZABLE, so using this option on a class that inherits from a non-serializable class will raise a compile-time error. The class types of any data members that are themselves class-based objects must also be SERIALIZABLE. The SERIALIZABLE option cannot be used with ABL-extended .NET classes.

When serializing an object to a JSON stream, the individual data members, properties, temp-tables, and ProDataSets of the class may also need to be defined as SERIALIZABLE themselves to participate in serialization. See the DEFINE VARIABLE statement, DEFINE PROPERTY statement, DEFINE TEMP-TABLE statement, and DEFINE DATASET statement entries for more information.

The conditions an object must satisfy to be passed between an application server and an ABL client are detailed in the Parameter passing syntax entry. The conditions an object must satisfy to be serialized to a stream are found in the relevant `Serialize( )` method entries, depending on the stream format.

class-body

The body of a class definition is composed of the following types of elements:

- **Data members** — Class members that define instance or static data (or the state) for the class
- **Properties** — Class members that define instance or static data, similar to data members, but with the option of associated behavior

- **Methods** — Class members that define instance or static behavior for the class
- **Class events** — Class members that define instance or static events for the class
- **Constructors** — Special methods that define initialization for the class, at least one of which is invoked for the class when an instance (object) is created, and a separate one of which is invoked the first time a class with static members is referenced
- **Destructor** — A special method that defines finalization behavior and that is invoked when the AVM deletes the object using garbage collection or when you delete the object explicitly
- **Class-scoped handle-based objects** — Handle-based objects that are not class members, but can be defined in and privately scoped to provide certain resources to the class
- **Triggers** — ON statements, which are not class members, but can be defined in the class to handle events on class-scoped and other handle-based objects
- **User-defined function prototypes** — FUNCTION statements that are not class members but declare external user-defined functions referenced by the class

You can define these elements in any order. Only elements that are listed as class members contribute to the type definition of a class. The remaining elements provide weakly typed resources that are privately available to the class definition.

Define elements in the class body using the following syntax:

```
[ data-member-definitions ]
[ property-definitions ]
[ method-definitions ]
[ event-definitions ]
[ constructor-definitions ]
[ destructor-definition ]
[ class-scoped-definitions ]
[ trigger-definitions ]
[ udf-prototype-definitions ]

END [ CLASS ] .
```

data-member-definitions

Defines one or more data members of the class. The following table lists the types of data members you can define and their associated ABL DEFINE statement. Data members can be defined as instance or static members. An instance data member is scoped to a particular class instance, while a static data member is scoped to the duration of the ABL session.

Table 13: Class data member types

Data member type	ABL DEFINE statement
BUFFER object	DEFINE BUFFER statement
Class-based object	DEFINE VARIABLE statement (with the AS CLASS option, defines a variable as an object reference to a class or interface type)

Data member type	ABL DEFINE statement
DATASET object	DEFINE DATASET statement
DATA-SOURCE object	DEFINE DATA-SOURCE statement
HANDLE	DEFINE VARIABLE statement (a primitive type that can reference a handle-based object)
Primitive type	DEFINE VARIABLE statement (for example, INTEGER or HANDLE)
QUERY object	DEFINE QUERY statement
TEMP-TABLE object	DEFINE TEMP-TABLE statement

For more information on these data member types, and any class-related restrictions, see the associated ABL DEFINE statement reference entry listed in the above table. For more information on accessing data members, see the Class-based data member access reference entry.

Note: If the data member is defined in a class that is derived from a .NET class, ABL imposes specific restrictions on the available names you can use to define data members. For more information, see the Notes of this reference entry.

---

property-definitions

Defines one or more instance or static property members of the class. If the class is abstract, an instance property can also be defined as abstract. Properties are similar to data members with the addition of behavioral and associated access control options. Like data members, an instance property is scoped to a particular class instance, while a static property is scoped to the duration of the ABL session. For more information on defining properties, see the DEFINE PROPERTY statement reference entry. For more information on accessing properties, see the Class-based property access reference entry.

Note: If the property is defined in a class that is derived from a .NET class, ABL imposes specific restrictions on the available names you can use to define properties. For more information, see the Notes of this reference entry.

---

method-definitions

Defines one or more instance or static method members of the class. If the class is abstract, an instance method can also be defined as abstract. A method is a named, callable block of ABL code that can return a value and that you can invoke like a procedure or a user-defined function (when it returns a value). You can define multiple overloaded methods with the same name, but with different parameter lists, and you can define a method that overrides or redefines a method of the same name and signature that is defined in a super class. Like data members and properties, an instance method is scoped to a particular class instance and a static method is scoped to the duration of the ABL session. For more information on defining methods, see the METHOD statement reference entry. For more information on calling methods, see the Class-based method call reference entry.

Note: If the method is defined in a class that is derived from a .NET class, ABL imposes specific restrictions on the available method names you can use to define methods. For more information, see the Notes of this reference entry.

event-definitions

Defines one or more instance or static event members of the class (class events). If the class is abstract, an instance event can also be defined as abstract. A class event allows the defining class to send notifications that some condition has been identified. Like data members, properties, and methods, an instance event is scoped to a particular class instance and a static event is scoped to the duration of the ABL session. For more information on defining class events, see the DEFINE EVENT statement reference entry. For more information on working with class events, see the Class Events Reference.

constructor-definitions

Defines one or more constructors for the class. Only one such constructor can be defined as static. An instance constructor is a special method that ABL invokes to initialize data for a new class instance when the object is instantiated using the NEW function (classes), NEW statement, or DYNAMIC-NEW statement. You can define multiple instance constructors distinguished (overloaded) by different parameter lists. You cannot invoke an instance constructor directly except from within the body of another constructor. If you do not define an instance constructor, ABL provides a default instance constructor that takes no parameters. A static constructor is a special method that ABL invokes to initialize static members defined by the class. You can define one static constructor that ABL invokes on the first reference to a class. If you do not define a static constructor, ABL provides a default static constructor. For more information on defining constructors, see the CONSTRUCTOR statement reference entry.

destructor-definition

Defines a destructor for the class. You can define only one destructor for a class. ABL invokes this destructor when the an instance of the class is deleted, either automatically during garbage collection or manually using the DELETE OBJECT statement. You cannot invoke a destructor directly. If not defined, ABL provides a default destructor. For more information on defining a destructor, see the DESTRUCTOR statement reference entry.

class-scoped-definitions

Defines class-scoped handle-based objects that are not data members, but provide certain resources that are privately scoped to the class definition. These resources include the statically defined widgets that support the OpenEdge GUI and certain statically defined handle-based objects, all of which are listed in the following table.

Table 14: Class-scoped handle-based objects

Data member type	ABL DEFINE statement
BROWSE widget	DEFINE BROWSE statement
BUTTON widget	DEFINE BUTTON statement
Data stream object	DEFINE STREAM statement
FRAME widget	DEFINE FRAME statement

Data member type	ABL DEFINE statement
IMAGE widget	DEFINE IMAGE statement
MENU widget	DEFINE MENU statement
RECTANGLE widget	DEFINE RECTANGLE statement
SUB-MENU widget	DEFINE SUB-MENU statement
WORK-TABLE object	DEFINE WORK-TABLE statement

For more information on defining these class-scoped objects, and any class-related restrictions, see the associated ABL DEFINE statement reference entry.

trigger-definitions

Defines one or more ON statements that specify triggers for events on certain handle-based objects, which can include some of the class-scoped objects listed in the pr. For more information on defining triggers, see the ON statement reference entry.

udf-prototype-definitions

Declares user-defined function prototypes for functions that are defined external to the class but referenced by methods and triggers defined in the class. For more information on defining user-defined function prototypes, see the FUNCTION statement reference entry.

END [CLASS]

Specifies the end of the class body definition. You must end the class body definition with the END statement.

Examples

The following samples include three different sets of class and procedure definitions that provide similar functionality in different ways. Each class defines or inherits a ProDataSet data member (`dsHighCustData`) and a public class method (`SetHighCustomerData( )`) that populates the ProDataSet with selected fields in the `Sports2020` database from both a `Customer` record and related `Invoice` records, where the `Customer` has the highest recorded balance in the database and also has related invoices. Each implementation of `SetHighCustomerData( )` also sets two public properties defined by its class (`HighCustBalance` and `HighCustNum`) to the highest database value for `Customer.Balance` and `Customer.CustNum`, and publishes a public class event defined by its class for each `Customer` record it encounters with related `Invoice` records. Each class defines or inherits additional data members to support its own instance of `dsHighCustData`, and also defines or inherits a public method (`GetHighCustomerData( )`) that returns `dsHighCustData` as an output parameter.

Each of the following sample class files or sets of class files implement this functionality:

1. **`r-CustObj.cls`** — Using instance members of an `r-CustObj` ABL class instance
2. **`r-CustObjStatic.cls`** — Using static members of the `r-CustObjStatic` ABL class type
3. **`r-CustObjAbstractImpl.cls` and `r-CustObjAbstract.cls`** — Using instance members of an `r-CustObjAbstractImpl` ABL class instance, some of which are abstract members inherited from the `r-CustObjAbstract` ABL abstract class

Each class or set of classes has an associated sample procedure that accesses the respective class and implements a similar application to display data stored by the class.

The following `r-CustObj.cls` sample class file shows many of the basic features of an ABL class definition. It defines only instance members and is defined as FINAL, meaning that it cannot be inherited by a subclass. This class sample also implicitly inherits from `Progress.Lang.Object`.

In this class, most of the data members, including the `dsHighCustData` ProDataSet and its supporting temp-tables, query, and data-sources, are defined as private, allowing access to its data primarily through its public properties and methods. However to demonstrate a public data member, the handle variable for `dsHighCustData` is defined as public. The class constructor attaches these data sources to the ProDataSet temp-table buffers and sets the handle variable (`hHighCustData`) for reference by the `SetHighCustomerData( )` method to fill and refill the ProDataSet. (An application might need to call this method multiple times to refill the ProDataSet as database `Customer.Balance` values change.)

r-CustObj.cls

```
CLASS r-CustObj FINAL:

  /* Public properties to return basic values for a customer with the
     highest balance */
  DEFINE PUBLIC PROPERTY HighCustBalance AS DECIMAL INITIAL 0.0 NO-UNDO
    GET.
    PRIVATE SET.
  DEFINE PUBLIC PROPERTY HighCustNum AS INTEGER INITIAL ? NO-UNDO
    GET.
    PRIVATE SET.

  /* Public event to notify about Customers with Invoices */
  DEFINE PUBLIC EVENT CustHasInvoices
    SIGNATURE VOID ( piCustNum AS INTEGER ).

  /* Public handle variable for the high customer ProDataSet */
  DEFINE PUBLIC VARIABLE hHighCustData AS HANDLE NO-UNDO.

  /* Private temp-tables for the high customer ProDataSet */
  DEFINE PRIVATE TEMP-TABLE ttCust NO-UNDO LIKE Customer.
  DEFINE PRIVATE TEMP-TABLE ttInv  NO-UNDO LIKE Invoice.

  /* Private ProDataSet for a single customer with the highest
     balance and its invoices */
  DEFINE PRIVATE DATASET dsHighCustData FOR ttCust, ttInv
    DATA-RELATION FOR ttCust, ttInv
      RELATION-FIELDS (ttCust.CustNum, ttInv.CustNum).

  /* Private query and data sources for the ProDataSet */
  DEFINE PRIVATE QUERY qCust FOR Customer.
  DEFINE PRIVATE DATA-SOURCE srcCust FOR QUERY qCust.
  DEFINE PRIVATE DATA-SOURCE srcInv  FOR Invoice.

  /* Constructor to initialize handles and attach data sources */
  CONSTRUCTOR r-CustObj ( ):
    hHighCustData = DATASET dsHighCustData:HANDLE.
    BUFFER ttCust:ATTACH-DATA-SOURCE( DATA-SOURCE srcCust:HANDLE ).
    BUFFER ttInv:ATTACH-DATA-SOURCE( DATA-SOURCE srcInv:HANDLE ).
  END CONSTRUCTOR.

/* Public method to get the current high customer data */
  METHOD PUBLIC VOID GetHighCustomerData
    ( OUTPUT DATASET dsHighCustData BIND ):
  END METHOD.

/* Public method to set (or reset) the current high customer data */
  METHOD PUBLIC VOID SetHighCustomerData ( ):
    hHighCustData:EMPTY-DATASET( ).
    FOR EACH Customer: /* Find Customer with highest balance */
```

```
      FIND FIRST Invoice WHERE Invoice.CustNum = Customer.CustNum NO-ERROR.
      IF AVAILABLE Invoice THEN DO:
        IF Customer.Balance > HighCustBalance THEN
          ASSIGN  HighCustBalance = Customer.Balance
                  HighCustNum     = Customer.CustNum.
        CustHasInvoices:Publish( Customer.CustNum ).
      END.
    END.
    QUERY qCust:QUERY-PREPARE("FOR EACH Customer "
      + "WHERE Customer.CustNum = " + STRING(HighCustNum) ).
    hHighCustData:FILL( ).
  END METHOD.

END CLASS.
```

To access the instance members of this class from outside its class definition, another class or procedure must instantiate the class and assign its object reference to a property or variable whose data type is the instantiated class type, for example:

```
DEFINE VARIABLE rObj AS CLASS r-CustObj NO-UNDO.
rObj = NEW r-CustObj( ) NO-ERROR.
```

This example uses the NEW statement to instantiate the class. You can also use the NEW function (classes) or New( ) method in an appropriate expression, or the DYNAMIC-NEW statement. You can then reference the public members of the class using the object reference, for example, calling a public instance method of the class:

```
rObj:SetHighCustomerData( ) NO-ERROR.
```

For more information on using object references, see the reference entry for a Class-based object reference.

To access instance members of a class from within the class definition or a derived class, you can typically reference the member without any prefix, like any local variable of a procedure or method. However, when the class member has the name of a reserved keyword, you must prefix the reference with THIS-OBJECT. For more information, see the reference entry for the THIS-OBJECT system reference.

The following `r-CustObjProc.p` sample procedure file displays data from an `r-CustObj` class instance as follows (with numbers corresponding to the numbered comments in the sample):

1. Defines a static ProDataSet object with a schema that is equivalent to the private `dsHighCustData` ProDataSet member defined by the `r-CustObj` class
2. Instantiates the `r-CustObj` class using the NEW statement, assigning its object reference to a variable (`rObj`) that the procedure uses to access this `r-CustObj` instance and all of its public members
3. Subscribes its internal procedure, `CustHasInvoices_Handler`, as a handler for the public `CustHasInvoices` event on the `r-CustObj` instance
4. Displays a message showing the default values of the `r-CustObj` public instance properties, `HighCustNum` and `HighCustBalance`
5. Calls the `SetHighCustomerData( )` method on the `r-CustObj` instance, which initializes the `r-CustObj` public properties and private ProDataSet and publishes the `CustHasInvoices` event, causing the `CustHasInvoices_Handler` procedure to execute and display a message indicating each `Customer` that has invoices

6. Displays a message showing the initialized values of the `r-CustObj` public instance properties, `HighCustNum` and `HighCustBalance`
7. Calls the `GetHighCustomerData( )` method on the `r-CustObj` instance to return its private ProDataSet member, `dsHighCustData`, as an output parameter, and storing the contents in its own equivalent ProDataSet object
8. Displays the contents of `dsHighCustData` in a down frame, showing fields from the `Customer` (`ttCust`) with the highest stored balance and related `Invoice` (`ttInv`) fields

r-CustObjProc.p

```
/* 1 */
DEFINE TEMP-TABLE ttCust NO-UNDO REFERENCE-ONLY LIKE Customer.
DEFINE TEMP-TABLE ttInv  NO-UNDO REFERENCE-ONLY LIKE Invoice.

DEFINE DATASET dsHighCustData REFERENCE-ONLY FOR ttCust, ttInv
  DATA-RELATION FOR ttCust, ttInv
    RELATION-FIELDS (ttCust.CustNum, ttInv.CustNum).

DEFINE VARIABLE rObj AS CLASS r-CustObj NO-UNDO.
/* 2 */
rObj = NEW r-CustObj( ) NO-ERROR.
/* 3 */
rObj:CustHasInvoices:Subscribe( "CustHasInvoices_Handler" ) NO-ERROR.
/* 4 */
MESSAGE "High Customer Number:" rObj:HighCustNum SKIP
        "High Customer Balance:" rObj:HighCustBalance VIEW-AS ALERT-BOX.
/* 5 */
rObj:SetHighCustomerData( ) NO-ERROR.
/* 6 */
MESSAGE "High Customer Number:" rObj:HighCustNum SKIP
        "High Customer Balance:" rObj:HighCustBalance VIEW-AS ALERT-BOX.
/* 7 */
rObj:GetHighCustomerData( OUTPUT DATASET dsHighCustData BIND ) NO-ERROR.
/* 8 */
CURRENT-WINDOW:WIDTH-CHARS = 90.

FOR EACH ttCust, EACH ttInv BREAK BY ttInv.CustNum:
  DISPLAY ttCust.CustNum WHEN FIRST-OF(ttInv.CustNum)
          ttCust.Name WHEN FIRST-OF(ttInv.CustNum)
          ttCust.Balance WHEN FIRST-OF(ttInv.CustNum)
          ttInv.InvoiceNum ttInv.Amount SKIP
    WITH FRAME A WIDTH 90 DOWN
         TITLE "Customer with highest stored balance" NO-ERROR.
END.

PROCEDURE CustHasInvoices_Handler:
  DEFINE INPUT PARAMETER pCustNum AS INTEGER.

  FIND FIRST Customer WHERE Customer.CustNum = pCustNum NO-ERROR.
  IF AVAILABLE Customer THEN
    MESSAGE "Customer" Customer.CustNum ('"' + Customer.Name + '"')
            "has a stored balance of" Customer.Balance
            "and also has Invoices."
            VIEW-AS ALERT-BOX.
END PROCEDURE.
```

The following `r-CustObjStatic.cls` sample class file provides functionality similar to `r-CustObj.cls`, but using static members instead. In addition, the class is not defined as FINAL. So, it can be inherited by an ABL subclass (not shown, here). This sample class also implicitly inherits from `Progress.Lang.Object`. Note that this class could include instance members as well. However, static members cannot reference instance members of the same class hierarchy; so, any instance members would have to support additional functionality, which could only be accessible using an instance of the class. The supported functionality, using static members, is available only as a function of the class type.

So, in this class, all the members are static. Most of the data members, including the `dsHighCustData` ProDataSet and its supporting temp-tables, buffers, query, and data-sources, are defined as protected, primarily allowing access to its data either by a subclass or through its public properties and methods from outside the class hierarchy. However to demonstrate a public data member, the handle variable for `dsHighCustData` is defined as public. The static class constructor (like the constructor in `r-CustObj.cls`) attaches these data sources to the ProDataSet temp-table buffers and sets the handle variable (`hHighCustData`) for reference by the `SetHighCustomerData( )` to fill and refill the ProDataSet.

As another difference from `r-CustObj.cls`, note the use of the alternate static buffers (`bHighCust`, `bCustomer`, and `bInvoice`), which allow other static members, such as the static query and method members, to access the database `Customer` and `Invoice` tables. The default buffers of database tables cannot be accessed from a static class member because ABL treats these buffers implicitly as instance members of the same class.

r-CustObjStatic.cls

```
CLASS r-CustObjStatic:
  /* Public static properties to return basic values for a customer with the
     highest balance */
  DEFINE PUBLIC STATIC PROPERTY HighCustBalance AS DECIMAL INITIAL 0.0 NO-UNDO
    GET.
    PROTECTED SET.
  DEFINE PUBLIC STATIC PROPERTY HighCustNum AS INTEGER INITIAL ? NO-UNDO
    GET.
    PROTECTED SET.

/* Public static event to notify about Customers with Invoices */
  DEFINE PUBLIC STATIC EVENT CustHasInvoices
    SIGNATURE VOID ( piCustNum AS INTEGER ).

  /* Public static handle variable for the high customer ProDataSet */
  DEFINE PUBLIC STATIC VARIABLE hHighCustData AS HANDLE NO-UNDO.

  /* Protected static temp-tables for the high customer ProDataSet */
  DEFINE PROTECTED STATIC TEMP-TABLE ttCust NO-UNDO LIKE Customer.
  DEFINE PROTECTED STATIC TEMP-TABLE ttInv  NO-UNDO LIKE Invoice.

  /* Protected static ProDataSet for a single customer with the highest
     balance and its invoices */
  DEFINE PROTECTED STATIC DATASET dsHighCustData FOR ttCust, ttInv
    DATA-RELATION FOR ttCust, ttInv
      RELATION-FIELDS (ttCust.CustNum, ttInv.CustNum).

  /* Protected static buffers for the ProDataSet query and data sources */
  DEFINE PROTECTED STATIC BUFFER bHighCust FOR Customer.
  DEFINE PROTECTED STATIC BUFFER bCustomer FOR Customer.
  DEFINE PROTECTED STATIC BUFFER bInvoice  FOR Invoice.

  /* Protected static query and data sources for the ProDataSet */
  DEFINE PROTECTED STATIC QUERY qCust FOR bCustomer.
  DEFINE PROTECTED STATIC DATA-SOURCE srcCust FOR QUERY qCust.
  DEFINE PROTECTED STATIC DATA-SOURCE srcInv  FOR bInvoice.

  /* Static constructor */
  CONSTRUCTOR STATIC r-CustObjStatic ( ):
    hHighCustData = DATASET dsHighCustData:HANDLE.
    BUFFER ttCust:ATTACH-DATA-SOURCE( DATA-SOURCE srcCust:HANDLE ).
    BUFFER ttInv:ATTACH-DATA-SOURCE( DATA-SOURCE srcInv:HANDLE ).
  END CONSTRUCTOR.

  /* Public static method to get the current high customer data */
  METHOD PUBLIC STATIC VOID GetHighCustomerData
    ( OUTPUT DATASET dsHighCustData BIND ):
  END METHOD.
  /* Public static method to set (or reset) the current high customer
     data */
```

```
  METHOD PUBLIC STATIC VOID SetHighCustomerData ( ):
    hHighCustData:EMPTY-DATASET( ).
    FOR EACH bHighCust: /* Find first customer with highest balance */
      FIND bInvoice WHERE bInvoice.CustNum = bHighCust.CustNum NO-ERROR.
      IF AVAILABLE bInvoice THEN DO:
        IF bHighCust.Balance > HighCustBalance THEN
          ASSIGN  HighCustBalance = bHighCust.Balance
                  HighCustNum     = bHighCust.CustNum.
        CustHasInvoices:Publish( bHighCust.CustNum ).
      END.
    END.
    QUERY qCust:QUERY-PREPARE("FOR EACH bCustomer "
      + "WHERE bCustomer.CustNum = " + STRING(HighCustNum) ).
    hHighCustData:FILL( ).
  END METHOD.

END CLASS.
```

To access the static members of this class from outside of its class definition (and any derived class), you do not need to instantiate the class. You only need to reference the member using static type-name syntax, for example:

```
r-CustObjStatic:SetHighCustomerData( ) NO-ERROR.
```

For more information on static type-name syntax, see the Type-name syntax, Class-based data member access, Class-based method call, and Class-based property access reference entries.

To access static members of a class from within the class definition or a derived class, you can typically reference the member without any prefix, like any local variable of a procedure or method. However, when the class member has the name of a reserved keyword, you must prefix the reference using static type-name syntax.

The following `r-CustObjStaticProc.p` sample procedure file displays static data from the `r-CustObjStatic` class type in a manner similar to how `r-CustObjProc.p` displays instance data from an `r-CustObj` class instance, except all references to public members are through the class type (`r-CustObjStatic`) instead of through an object reference (`rObj` in `r-CustObjProc.p`). Thus, `r-CustObjStaticProc.p` has no need to instantiate `r-CustObjStatic`. Otherwise, the application is exactly the same.

r-CustObjStaticProc.p

```
DEFINE TEMP-TABLE ttCust NO-UNDO REFERENCE-ONLY LIKE Customer.
DEFINE TEMP-TABLE ttInv  NO-UNDO REFERENCE-ONLY LIKE Invoice.

DEFINE DATASET dsHighCustData REFERENCE-ONLY FOR ttCust, ttInv
  DATA-RELATION FOR ttCust, ttInv
      RELATION-FIELDS (ttCust.CustNum, ttInv.CustNum).

r-CustObjStatic:CustHasInvoices:Subscribe( "CustHasInvoices_Handler" )
  NO-ERROR.

MESSAGE "High Customer Number:" r-CustObjStatic:HighCustNum SKIP
        "High Customer Balance:" r-CustObjStatic:HighCustBalance VIEW-AS ALERT-BOX.

r-CustObjStatic:SetHighCustomerData( ) NO-ERROR.
MESSAGE "High Customer Number:" r-CustObjStatic:HighCustNum SKIP
        "High Customer Balance:" r-CustObjStatic:HighCustBalance VIEW-AS ALERT-BOX.

r-CustObjStatic:GetHighCustomerData( OUTPUT DATASET dsHighCustData
```

```
    BIND )
 NO-ERROR.

CURRENT-WINDOW:WIDTH-CHARS = 90.

FOR EACH ttCust, EACH ttInv BREAK BY ttInv.CustNum:
  DISPLAY ttCust.CustNum WHEN FIRST-OF(ttInv.CustNum)
          ttCust.Name WHEN FIRST-OF(ttInv.CustNum)
          ttCust.Balance WHEN FIRST-OF(ttInv.CustNum)
          ttInv.InvoiceNum ttInv.Amount SKIP
    WITH FRAME A WIDTH 90 DOWN
         TITLE "Customer with highest stored balance" NO-ERROR.
END.

PROCEDURE CustHasInvoices_Handler:
  DEFINE INPUT PARAMETER pCustNum AS INTEGER.

  FIND FIRST Customer WHERE Customer.CustNum = pCustNum NO-ERROR.
  IF AVAILABLE Customer THEN
    MESSAGE "Customer" Customer.CustNum ('"' + Customer.Name + '"')
            "has a stored balance of" Customer.Balance
            "and also has Invoices."
            VIEW-AS ALERT-BOX.

END PROCEDURE.
```

The following sample class files, `r-CustObjAbstract.cls` and `r-CustObjAbstractImpl.cls`, provide functionality similar to `r-CustObj.cls`, but using a combination of abstract and non-abstract instance members that are defined in the abstract class, `r-CustObjAbstract`, and inherited by the non-abstract class, `r-CustObjAbstractImpl`.

The `r-CustObjAbstract.cls` class file defines the `r-CustObjAbstract` abstract class. Its abstract members consist of most of the public members of the class, including properties, methods, and the class event. This class also implicitly inherits from `Progress.Lang.Object`.

In this class, all the data members (which are always non-abstract), including the `dsHighCustData` ProDataSet and its supporting handle variable, temp-tables, buffers, query, and data-sources, are defined as protected, allowing direct access to its data from any subclass that inherits from it or through its public properties and methods (once implemented) from outside the class hierarchy. The abstract class constructor (like the constructor in `r-CustObj.cls`) attaches the data sources to the ProDataSet temp-table buffers and sets the handle variable (`hHighCustData`) for reference by the subclass that implements the abstract `SetHighCustomerData( )` method to fill and refill the ProDataSet. This constructor also subscribes a private method (`CustHasInvoices_Handler( )`) as a handler for the abstract `CustHasInvoices` event. This handler thus always responds to the event, no matter how it is implemented. Note that the class cannot publish the event, because it has not yet been implemented.

One public method, `GetHighCustomerData( )`, is implemented (not abstract) because its only function is to return `dsHighCustData` as an output parameter. If necessary, an abstract subclass can still override it again as abstract for implementation further down the class hierarchy, or it can simply be overridden by any subclass. The abstract method, `SetHighCustomerData( )`, is intended to be implemented any number of ways. For example, an alternative implementation could accumulate an actual `Customer` balance from related `Invoice.Amount` values instead of using the stored value of `Customer.Balance`, which does not necessarily match this total. For an example of this implementation, see the `r-ICustObjectImpl2.cls` sample class file, which is described in the Examples section of the INTERFACE statement reference entry.

Also, note the use of the protected alternate buffers (`bCustomer` and `bInvoice`). These buffers allow access to the database `Customer` and `Invoice` tables by certain protected class member definitions, including the ProDataSet query and data-sources, that cannot reference the default buffers of database tables. ABL implicitly treats the default buffers of database tables as private instance members of any class that references them; so they cannot be inherited along with any protected member definitions where they might be included. As such, the protected buffers allow any derived class to access the same buffers that are referenced in the protected member definitions it inherits.

r-CustObjAbstract.cls

```
CLASS r-CustObjAbstract ABSTRACT:

  /* Public abstract properties to return basic values for a customer
     with the highest balance */
  DEFINE PUBLIC ABSTRACT PROPERTY HighCustBalance AS DECIMAL NO-UNDO
    GET.
    PROTECTED SET.
  DEFINE PUBLIC ABSTRACT PROPERTY HighCustNum AS INTEGER NO-UNDO
    GET.
    PROTECTED SET.

  /* Public abstract event to notify about Customers with Invoices */
  DEFINE PUBLIC ABSTRACT EVENT CustHasInvoices
    SIGNATURE VOID ( piCustNum AS INTEGER ).

  /* Protected handle variable for the high customer ProDataSet */
  DEFINE PROTECTED VARIABLE hHighCustData AS HANDLE NO-UNDO.

  /* Protected temp-tables for the high customer ProDataSet */
  DEFINE PROTECTED TEMP-TABLE ttCust NO-UNDO LIKE Customer.
  DEFINE PROTECTED TEMP-TABLE ttInv  NO-UNDO LIKE Invoice.

  /* Protected ProDataSet for a single customer with the highest
     balance and its invoices */
  DEFINE PROTECTED DATASET dsHighCustData FOR ttCust, ttInv
    DATA-RELATION FOR ttCust, ttInv
      RELATION-FIELDS (ttCust.CustNum, ttInv.CustNum).

  /* Protected buffers for the ProDataSet query and data sources */
  DEFINE PROTECTED BUFFER bCustomer FOR Customer.
  DEFINE PROTECTED BUFFER bInvoice  FOR Invoice.

  /* Private query and data sources for the ProDataSet */
  DEFINE PROTECTED QUERY qCust FOR bCustomer.
  DEFINE PROTECTED DATA-SOURCE srcCust FOR QUERY qCust.
  DEFINE PROTECTED DATA-SOURCE srcInv  FOR bInvoice.

 /* Constructor to initialize handles and attach data sources */
  CONSTRUCTOR r-CustObjAbstract ( ):

    hHighCustData = DATASET dsHighCustData:HANDLE.
    BUFFER ttCust:ATTACH-DATA-SOURCE( DATA-SOURCE srcCust:HANDLE ).
    BUFFER ttInv:ATTACH-DATA-SOURCE( DATA-SOURCE srcInv:HANDLE ).
    CustHasInvoices:Subscribe( CustHasInvoices_Handler ) NO-ERROR.

  END CONSTRUCTOR.

  /* Default private event handler for the abstract CustHasInvoices event */
  METHOD PRIVATE VOID CustHasInvoices_Handler ( INPUT pArg AS INTEGER ):
    MESSAGE "Customer" pArg "has Invoices." VIEW-AS ALERT-BOX.
  END METHOD.

  /* Public method to get the current high customer data */
  METHOD PUBLIC VOID GetHighCustomerData
    ( OUTPUT DATASET FOR dsHighCustData BIND ):

  END METHOD.
```

```
  /* Public abstract method to set (or reset) the current high customer
     data */
  METHOD PUBLIC ABSTRACT VOID SetHighCustomerData ( ).

END CLASS.
```

The `r-CustObjAbstractImpl.cls` class file defines the non-abstract `r-CustObjAbstractImpl` class, which inherits `r-CustObjAbstract` and implements its abstract members. The class is not defined as FINAL. So, it can be inherited by an ABL subclass (not shown, here).

The abstract member implementations include initial values for the `HighCustBalance` and `HighCustNum` properties, the `CustHasInvoices` event so it can be published, and the `SetHighCustomerData( )` method, which is implemented almost exactly the same as for `r-CustObj`, to set these properties, fill the ProDataSet, and publish the event when appropriate.

r-CustObjAbstractImpl.cls

```
CLASS r-CustObjAbstractImpl INHERITS r-CustObjAbstract:

  /* Public properties implemented to return basic values for a customer
     with the highest balance */
  DEFINE OVERRIDE PUBLIC PROPERTY HighCustBalance AS DECIMAL INITIAL 0.0
      NO-UNDO
    GET.
    PROTECTED SET.
  DEFINE OVERRIDE PUBLIC PROPERTY HighCustNum AS INTEGER INITIAL ?
      NO-UNDO
    GET.
    PROTECTED SET.

  /* Public event implemented to notify about Customers with Invoices */
  DEFINE PUBLIC OVERRIDE EVENT CustHasInvoices
    SIGNATURE VOID (INPUT piCustNum AS INTEGER).

  /* Public method implemented to set (or reset) the current high
     customer data based on the stored value of Customer.Balance */
  METHOD OVERRIDE PUBLIC VOID SetHighCustomerData ( ):

    hHighCustData:EMPTY-DATASET( ).
    FOR EACH Customer: /* Find Customer with highest balance */
      FIND FIRST Invoice WHERE Invoice.CustNum = Customer.CustNum NO-ERROR.
      IF AVAILABLE Invoice THEN DO:
        IF Customer.Balance > HighCustBalance THEN
          ASSIGN  HighCustBalance = Customer.Balance
                  HighCustNum     = Customer.CustNum.
        CustHasInvoices:Publish( Customer.CustNum ).
      END.
    END.
    QUERY qCust:QUERY-PREPARE("FOR EACH bCustomer "
      + "WHERE bCustomer.CustNum = " + STRING(HighCustNum) ).
    hHighCustData:FILL( ).

  END METHOD.

END CLASS.
```

Note that from within r-CustObjAbstractImpl, members inherited from r-CustObjAbstract (such as hHighCustData and HighCustBalance) are accessed without any prefix, as if they were defined in the same class. You can access any inherited member of a super class this way. However, if an inherited instance member has the name of a reserved keyword, you must prefix the member reference with THIS-OBJECT. For more information, see the reference entry for the THIS-OBJECT system reference. For an inherited static member named with a reserved keyword, you must use static type-name syntax as previously described for the r-CustObjStatic class.

The following r-CustObjAbstractProc.p sample procedure file displays data from an instance of r-CustObjAbstractImpl in a manner similar to how r-CustObjProc.p displays data from an instance of r-CustObj. The only difference is in the event handler procedure (CustHasInvoices_Handler) which responds to the CustHasInvoice event in way that works more smoothly with the event handler provided by the r-CustObjAbstract class. Otherwise, the application is exactly the same.

r-CustObjAbstractProc.p

```
DEFINE TEMP-TABLE ttCust NO-UNDO REFERENCE-ONLY LIKE Customer.
DEFINE TEMP-TABLE ttInv  NO-UNDO REFERENCE-ONLY LIKE Invoice.

DEFINE DATASET dsHighCustData REFERENCE-ONLY FOR ttCust, ttInv
  DATA-RELATION FOR ttCust, ttInv
    RELATION-FIELDS (ttCust.CustNum, ttInv.CustNum).

DEFINE VARIABLE rObj AS CLASS r-CustObjAbstractImpl NO-UNDO.

rObj = NEW r-CustObjAbstractImpl( ) NO-ERROR.
rObj:CustHasInvoices:Subscribe( "CustHasInvoices_Handler" ) NO-ERROR.

MESSAGE "High Customer Number:" rObj:HighCustNum SKIP
        "High Customer Balance:" rObj:HighCustBalance VIEW-AS ALERT-BOX.

rObj:SetHighCustomerData( ) NO-ERROR.

MESSAGE "High Customer Number:" rObj:HighCustNum SKIP
        "High Customer Balance:" rObj:HighCustBalance VIEW-AS ALERT-BOX.

rObj:GetHighCustomerData( OUTPUT DATASET dsHighCustData BIND ) NO-ERROR.

CURRENT-WINDOW:WIDTH-CHARS = 90.

FOR EACH ttCust, EACH ttInv BREAK BY ttInv.CustNum:
  DISPLAY ttCust.CustNum WHEN FIRST-OF(ttInv.CustNum)
          ttCust.Name WHEN FIRST-OF(ttInv.CustNum)
          ttCust.Balance WHEN FIRST-OF(ttInv.CustNum)
          ttInv.InvoiceNum ttInv.Amount SKIP
    WITH FRAME A WIDTH 90 DOWN
         TITLE "Customer with highest stored balance" NO-ERROR.
END.

PROCEDURE CustHasInvoices_Handler:
  DEFINE INPUT PARAMETER pCustNum AS INTEGER.

  FIND FIRST Customer WHERE Customer.CustNum = pCustNum NO-ERROR.
  IF AVAILABLE Customer THEN
    MESSAGE pCustNum ('"' + Customer.Name + '"')
            "has a stored balance of" (STRING(Customer.Balance) + ".")
            VIEW-AS ALERT-BOX.

END PROCEDURE.
```

For an example of an ABL class that inherits from a .NET class, see the WAIT-FOR statement (.NET and ABL) reference entry.

Notes

- You can terminate a CLASS statement with either a period (.) or a colon (:), but typically use a colon (:).
- A class definition (`.cls`) file can contain only one class definition that is optionally preceded by one or more USING statements, a BLOCK-LEVEL ON ERROR UNDO, THROW statement, or a ROUTINE-LEVEL ON ERROR UNDO, THROW statement. The complete class definition must begin with the CLASS statement and end with the END statement, and the CLASS statement must be the first compilable statement after any USING. BLOCK-LEVEL ON ERROR UNDO, THROW, or ON ERROR UNDO, THROW statements in the file. A class definition file containing a class definition cannot also contain an interface definition.
- The access mode for a class definition is always PUBLIC.
- In effect, a user-defined class represents a unique data type. In ABL, you can use a class type much as you would any ABL built-in data type. You can define variables, parameters, return types, and class-based properties as a class type. These data elements can then hold a reference to a class instance (*object reference*). You can also assign an object reference to a temp-table field defined as the Progress.Lang.Object class type; but you cannot assign an object reference to a database table field. You can use the object reference to a class to access PUBLIC instance members of that class. For more information on object references, see the reference entry for a Class-based object reference. You can also use a class type name to access available static members of that class, whether or not an instance of the class exists. For more information on using class type names to access static class members, see the Class-based data member access, Class-based method call, Class-based property access, and Type-name syntax reference entries.
- The class name part of *`class-type-name`* can be an ABL reserved keyword (such as Display). If it is a reserved keyword, note that ABL does not fully support user-defined class names that are identical to reserved keywords. For more information, see the Type-name syntax reference entry.
- You can reference include files from within a class definition file. For more information about include files, see the reference entry for an { } Include file reference.
- All built-in preprocessor directives are supported in class definition files.
- All built-in preprocessor names are supported in class definition files. For a list of preprocessor names, see the reference entry for an { } Preprocessor name reference.
- You cannot pass compile-time arguments to class definition files. However, you can pass compile-time arguments to include files referenced in a class definition file.
- The compiled version of a class definition file is an r-code (`.r`) file. For more information, see the COMPILE statement reference entry.
- You cannot run r-code compiled for a class definition file with the RUN statement.
- Non-PRIVATE data members and properties within a class definition (`.cls`) file maintain their characteristics throughout the inherited class hierarchy. Thus, you cannot override non-PRIVATE data members and properties in a subclass that are defined in a super class. In other words, you cannot define a data member or property in a subclass using the same name as a non-PRIVATE data member or property defined in one of its super classes.
- You can create an instance of a class using the NEW function, the `New( )` method of the `Progress.Lang.Class` class, the NEW statement, or the DYNAMIC-NEW statement, and assign the object reference returned for that instance as the value of a data element defined to reference instances of that class type. You can access an object instance, as well as its PUBLIC instance data members, properties, and methods, using its associated object reference. PACKAGE-PROTECTED and PACKAGE-PRIVATE instance data members, properties, and methods can be accessed by objects within the same package as the object instance. For more information on instantiating classes as objects, see the CONSTRUCTOR statement, NEW function (classes), New( ) method of the Progress.Lang.Class class, NEW statement, or DYNAMIC-NEW statement reference entries. For more information on referencing class instances, see the Class-based object reference entry.

- The static members of a class are initialized with the first reference to its class type and just prior to execution of its static constructor, whether or not the class is instantiated in the process. These members remain in ABL session memory until the session ends or until the defining class is recompiled. Note that static data members and properties are not constant; the values of static data members can be changed like any data member, and the values of static properties can be changed as provided by the property definition. For more information on static member initialization, see the CONSTRUCTOR statement reference entry.
- ABL provides a system reference for the currently running instance of this class, called THIS-OBJECT. For more information, see the reference entry for the THIS-OBJECT system reference.
- If this class is a subclass of some super class, you can use the SUPER system reference within a class instance to access the PUBLIC, PACKAGE-PROTECTED, and PROTECTED instance methods of all super classes within the inherited class hierarchy. For more information, see the reference entry for the SUPER system reference. You can also use static type-name syntax to access any static method that you redefine (override) in the class hierarchy. For more information, see the Type-name syntax reference entry.
- You can store class definition r-code files in ABL procedure libraries. If the AVM encounters a procedure library on PROPATH, it will search the library for the specified r-code. However, you cannot execute r-code files stored in a procedure library that is not on `PROPATH` using the *`procedure-library-path`*`<<`*`member-name`*`>>` syntax.
- You cannot define a NEW SHARED or NEW GLOBAL SHARED variable in a class definition (`.cls`) file.
- You cannot use ActiveX controls within a class definition (`.cls`) file. However, you can use COM automation objects within a class definition (`.cls`) file, with any event handlers for a COM object defined in a procedure file.
- For more information on class definition (`.cls`) files, see *Develop Object-oriented ABL Applications*.
- If you inherit a .NET class, implement a .NET interface, or otherwise access a .NET object type in a class definition, ABL must find the assembly that defines the .NET object type at compile time. For information on how ABL locates and identifies .NET object types, see the Type-name syntax reference entry.
- Although you can override the .NET `WndProc( )` method, doing so can be risky. The method will be called for every event, like `WM_MOUSEMOVE` and `WM_PAINT`, so it could be called thousands of times for a single form, causing performance issues. It can also be called at unexpected times and cause unexpected behavior. Overriding WndProc when using ABL embedded windows with .NET forms could be especially problematic. See *Develop Object-oriented ABL Applications* for more information.
- If you inherit a .NET class, the `GetType( )` instance method (inherited from `System.Object`) returns incomplete information for the ABL part of the object.
- If you inherit a .NET class, you **cannot**:
  - Override the .NET Dispose( ) or Finalize( ) method
  - Define a method with certain reserved method names

  For more information, see the METHOD statement reference entry.
- You **cannot**:
  - Inherit from a .NET class that is, itself, defined with any of the following .NET directives (as specified in C#):
    - `internal`
    - `private`
    - `sealed`
    - `static`
  - Define an ABL class that can become an additional part for a .NET `partial` class

- Inherit from a .NET abstract class that defines an abstract indexed property
- Inherit from a .NET generic class
- Inherit from the following .NET class types or any class types that are derived from them:
  - `System.Delegate`
  - `System.Enum`
  - `System.Threading.Thread`
  - `System.ValueType`
- Implement a .NET interface that:
  - .NET defines as `private` or `internal` (as specified in C#)
  - Is generic or that defines a generic method prototype
  - Defines an indexed property prototype (including a default indexed property)

See also

Assignment (=) statement, BLOCK-LEVEL ON ERROR UNDO, THROW statement, Class-based object reference, CONSTRUCTOR statement, DEFINE EVENT statement, DEFINE PROPERTY statement, DEFINE VARIABLE statement, DESTRUCTOR statement, DYNAMIC-NEW statement, FUNCTION statement, INTERFACE statement, METHOD statement, NEW function (classes), New( ) method, NEW statement, ON statement, ROUTINE-LEVEL ON ERROR UNDO, THROW statement, Statements defining other class elements as specified in Table 13: Class data member types and Table 14: Class-scoped handle-based objects, Type-name syntax, USING statement, VAR statement

CLEAR statement

Clears the data for all fill-in fields in a frame. It also clears the colors for all widgets in a frame, except for enabled fill-ins.

Note: Does not apply to SpeedScript programming.

Syntax

```
CLEAR [ FRAME frame ] [ ALL ] [ NO-PAUSE ]
```

`FRAME` *`frame`*

Represents the name of the frame containing the fill-in fields you want to clear. If you do not name a frame, CLEAR clears the default frame for the block containing the CLEAR statement.

`ALL`

Clears all occurrences and resets the current display position to the top of the frame for a down frame (a frame used to display several occurrences of the fields in the frame).

NO-PAUSE

Does not pause before clearing the frame.

Example

The `r-clear.p` procedure displays the ABL data types and their corresponding default formats. The procedure prompts you to enter values so you can see how ABL formats those values. If you answer YES, the AVM clears the values currently displayed so that you can enter new values.

r-clear.p

```
DEFINE VARIABLE a AS CHARACTER NO-UNDO INITIAL "xxxxxxxx".
DEFINE VARIABLE b AS DATE      NO-UNDO INITIAL TODAY.
DEFINE VARIABLE c AS DECIMAL   NO-UNDO INITIAL "-12,345.67".
DEFINE VARIABLE d AS INTEGER   NO-UNDO INITIAL "-1,234,567".
DEFINE VARIABLE e AS LOGICAL   NO-UNDO INITIAL TRUE.

DISPLAY "This illustrates the default formats for the different data types"
  SKIP (2) WITH CENTERED ROW 4 NO-BOX FRAME head.
FORM "CHARACTER default format is ""x(8)"" " a SKIP
  "DATE default format is 99/99/99         " b SKIP
  "DECIMAL default format is ->>,>>9.99    " c SKIP
  "INTEGER default format is ->,>>>,>>9    " d SKIP
  "LOGICAL default format is yes/no        " e TO 55 SKIP
  WITH ROW 8 NO-BOX NO-LABELS CENTERED FRAME ex.

REPEAT:
  DISPLAY a b c d WITH FRAME ex.
  MESSAGE "Do you want to put in some values?"
  UPDATE e.
  IF e THEN DO:
    CLEAR FRAME ex NO-PAUSE.
    SET a b c d WITH FRAME ex.
  END.
  ELSE LEAVE.
END.
```

Notes

- The CLEAR statement only clears fill-in fields. GUI widgets such as editors or radio-sets are not affected.
- The AVM automatically clears a single (1 down) frame whenever its block is iterated, and automatically clears a multi-frame (down frame) whenever it is full and its block is iterated.
- When clearing colors for a field-level widget in a frame, the CLEAR FRAME statement honors all widget attribute and option settings that specify how the widget inherits foreground or background colors from its containing frame.
- To clear the data from a single fill-in without clearing its colors, use the CLEAR( ) method (FILL-IN Widget).

CLOSE QUERY statement

Closes a query that was opened by a previous OPEN QUERY statement.

Syntax

```
CLOSE QUERY query
```

query

The name of an open query.

Example

The `r-clsqry.p` procedure defines a query, q-cust, which it shares with `r-query.p`. Each time you choose the Ascending, Descending, or CustNum button, the procedure opens a new query for q-cust. To do this, the procedure must first close an open query for each q-cust. Therefore, the CLOSE QUERY statement is used in the CHOOSE trigger for each of these buttons.

r-clsqry.p

```
DEFINE NEW SHARED BUFFER x-cust FOR Customer.
DEFINE NEW SHARED QUERY  q-cust FOR x-cust.

DEFINE BUTTON b_quit LABEL "Quit"
  TRIGGERS:
    ON CHOOSE QUIT.
  END.

DEFINE BUTTON b_ascend  LABEL "Ascending".
DEFINE BUTTON b_descend LABEL "Descending".
DEFINE BUTTON b_num     LABEL "CustNum".

FORM b_ascend b_descend b_num b_quit
  WITH FRAME butt-frame ROW 1.

ON CHOOSE OF b_ascend DO:
  CLOSE QUERY q-cust.
  OPEN QUERY q-cust FOR EACH x-cust NO-LOCK BY x-cust.name.
  DISABLE ALL WITH FRAME butt-frame.
  RUN r-query.p.
END.

ON CHOOSE OF b_descend DO:
  CLOSE QUERY q-cust.
  OPEN QUERY q-cust FOR EACH x-cust NO-LOCK
    BY x-cust.name DESCENDING.
  DISABLE ALL WITH FRAME butt-frame.
  RUN r-query.p.
END.

ON CHOOSE OF b_num DO:
  CLOSE QUERY q-cust.
  OPEN QUERY q-cust FOR EACH x-cust NO-LOCK
    BY x-cust.CustNum.
  DISABLE ALL WITH FRAME butt-frame.
  RUN r-query.p.
END.

DO WHILE TRUE:
  ENABLE ALL WITH FRAME butt-frame.
  WAIT-FOR CHOOSE OF b_ascend, b_descend, b_num, b_quit.
END.
```

r-query.p

```
DEFINE SHARED BUFFER x-cust FOR Customer.
DEFINE SHARED QUERY  q-cust FOR x-cust.

GET FIRST q-cust.

DO WHILE AVAILABLE(x-cust):
  DISPLAY x-cust.name x-cust.custnum
    WITH FRAME cust-info CENTERED DOWN ROW 3 USE-TEXT.
  DOWN 1 WITH FRAME cust-info.
  GET NEXT q-cust.
END.
```

Notes

- If a query is closed, you cannot retrieve any more records for the query.
- Closing a query frees most resources used by the query.
- After you close a query, you can reopen it with the OPEN QUERY statement. However, you cannot reuse the query's buffers for a different table. For example, a buffer, buff1, is created for the customer table in a DEFINE QUERY or OPEN QUERY for the query, qry1. The query is run and closed. You cannot now DEFINE or OPEN qry1 with buff1 for the item table. You can reuse buffers with CREATE QUERY, but you must re-run QUERY-PREPARE.
- If you do not explicitly close a query, it is closed when another OPEN QUERY statement is executed for the same query name.

See also

CURRENT-RESULT-ROW function, DEFINE QUERY statement, GET statement, NUM-RESULTS function, OPEN QUERY statement, REPOSITION statement

CLOSE STORED-PROCEDURE statement

For a non-ABL stored procedure, indicates that the procedure has completed execution and retrieves any return status. For a send-sql-statement stored procedure, closes the SQL cursor used by the procedure.

Syntax

```
CLOSE STORED-PROCEDURE procedure
   [ integer-field = PROC-STATUS ]
   [ WHERE PROC-HANDLE = integer-field ]
```

procedure

The name of the stored procedure that you want to close or the built-in procedure name, send-sql-statement.

integer-field = PROC-STATUS

Assigns the return value from a stored procedure to the specified integer field or variable (*integer-field*).

WHERE PROC-HANDLE = *integer-field*

An integer field or variable whose value uniquely identifies the stored procedure that produces the results returned from the data source or the SQL cursor of a send-sql-statement stored procedure.

Example

The PROC-STATUS clause of the CLOSE STORED-PROCEDURE statement allows the DataServer for ORACLE to retrieve the text of an ORACLE error message that was passed to raise_application_error. Use the ERROR-STATUS:GET-MESSAGE handle to retrieve the message, as shown in the following example:

```
DEFINE VARIABLE st AS INTEGER NO-UNDO.
DEFINE VARIABLE h  AS INTEGER NO-UNDO.

RUN STORED-PROC p1 h = PROC-HANDLE NO-ERROR.

CLOSE STORED-PROC p1 st = PROC-STATUS WHERE PROC-HANDLE = h.

DISPLAY st.

IF ERROR-STATUS:ERROR THEN
  MESSAGE ERROR-STATUS:GET-MESSAGE(1) ERROR-STATUS:GET-NUMBER(1)
    VIEW-AS ALERT-BOX.
```

Notes

- If you specified a PROC-HANDLE when you ran a stored procedure, you must specify the PROC-HANDLE when you close the stored procedure.
- If you do not specify a PROC-HANDLE, the CLOSE STORED-PROCEDURE statement will close the procedure if there is only one stored procedure running. If there is more than one stored procedure running, an error will be returned.
- You cannot close a send-sql-statement procedure until you have retrieved all row results.
- You can close all stored procedures at once with the following statement:

```
RUN STORED-PROC closeallprocs.
```

- For more information on using this statement, see *Use the Oracle Data Server*.

See also

PROC-HANDLE function, PROC-STATUS function, RUN STORED-PROCEDURE statement

CODEPAGE-CONVERT function

Converts a string value from one code page to another.

Syntax

```
CODEPAGE-CONVERT
  ( source-string
    [ , target-codepage  [ ,source-codepage ] ]
  )
```

source-string

A CHARACTER or LONGCHAR expression to be converted.

target-codepage

A character-string expression that evaluates to the name of a code page. The returned character value is relative to *target-codepage*. The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that ABL uses for character management). If you supply a non-valid name, the CODEPAGE-CONVERT function returns the Unknown value (?). Before returning a character value, the CODEPAGE-CONVERT function converts *source-string* from *source-codepage* to *target-codepage*. If you do not specify *target-codepage*, no code page conversions occur.

source-codepage

A character-string expression that evaluates to the name of a code page. The *source-codepage* specifies the name of the code page to which *source-string* is relative. The name that you specify must be a valid code page name available in the DLC/convmap.cp file. If you supply a non-valid name, the CODEPAGE-CONVERT function returns the Unknown value (?). The default value of *source-codepage* is the value of CHARSET attribute of the SESSION handle.

If *source-string* is a LONGCHAR variable, the *source-codepage* argument is not valid. In this case, the code page of the LONGCHAR variable is used as the source code page.

Example

This example assumes that the native code page of r-codpag.p is ibm850. It is written so that its embedded text strings are always converted to the internal code page of the ABL session (SESSION:CHARSET).

r-codpag.p

```
DEFINE VARIABLE cp850string   AS CHARACTER NO-UNDO
  INITIAL "text with umlaut (ä)".
DEFINE VARIABLE charsetstring AS CHARACTER NO-UNDO.

charsetstring = CODEPAGE-CONVERT(cp850string, SESSION:CHARSET, "ibm850").

FOR EACH Item NO-LOCK:
  IF LOOKUP(charsetstring, Item.CatDescription) > 0 THEN
    DISPLAY Item.ItemName.
END.
```

Notes

- The CODEPAGE-CONVERT function returns the corresponding character string in the specified code page. By default, the value of SESSION:CHARSET is iso8859-1. You can set a different internal code page by specifying the Internal Code Page (-cpinternal) parameter. For more information, see *Internationalize ABL Applications* and *Startup Command and Parameter Reference*.

- This function is especially useful if you plan to run a procedure in an ABL session in which the SESSION:CHARSET code page is different from the native code page of the procedure.
- When you write procedures with ABL, you must use 7-bit (that is, ASCII) characters for field names and variable names. But you can use 8-bit and multi-byte characters, including Unicode, for data values such as character strings and constants. Thus, a procedure written and compiled on a system using one code page can be run on a system using another code page as long as you convert all embedded character strings to the internal code page. Using CODEPAGE-CONVERT as shown in the example allows your procedures to be virtually code page independent.

See also

ASC function, CHR function, STRING function

COLOR phrase

Specifies a video attribute or color. The COLOR phrase is superseded by the FGCOLOR and BGCOLOR options in graphical user interfaces and by the PFCOLOR and DCOLOR options in character interfaces. The COLOR phrase is supported only for backward compatibility with Progress Version 7 or earlier.

Note: Does not apply to SpeedScript programming.

Syntax

```
{    NORMAL
   | INPUT
   | MESSAGES
   | protermcap-attribute
   | dos-hex-attribute
   | { [ BLINK- ] [ BRIGHT- ]
       [ fgnd-color ] [ bgnd-color ]
     }
   | { [ BLINK- ] [ RVV- ] [ UNDERLINE- ] [ BRIGHT- ]
       [ fgnd-color ]
     }
   | VALUE ( expression )
}
```

`NORMAL, INPUT, MESSAGES`

The three standard colors ABL uses for screen displays. ABL uses NORMAL to display fields, INPUT to display input fields, and MESSAGES to display items in the message area.

Following are the NORMAL defaults:

- **Windows** — On a color monitor, the default colors are a blue background and a white foreground. On a monochrome monitor, the default colors are a standard background and foreground, depending on the monitor.

- **UNIX** — The default colors are the normal display mode of your terminal.

Following are the INPUT defaults:

- **Windows** — On a color monitor, the default colors are a light gray background and a blue foreground. On a monochrome monitor, the default underlines fields that require input.
- **UNIX** — The default colors depend on the type of terminal and how INPUT is defined in the protermcap file, but it is usually underlining.

Following are the MESSAGES defaults:

- **Windows** — On a color monitor, the defaults are the same as for INPUT. On a monochrome monitor, the default is reverse video.
- **UNIX** — The defaults depend on the type of terminal and how MESSAGES is defined in the protermcap file, but it is usually reverse video. (The protermcap file supplied with ABL supplies default attributes for NORMAL, INPUT, and MESSAGES for all defined terminals.)

protermcap-attribute

You use the *protermcap-attribute* option only if you are using UNIX. This is the name assigned to the attribute in the *protermcap* file (for example, RED, BLINK, etc.). See *Manage ABL Applications* for a description of the *protermcap* file.

dos-hex-attribute

A hex string with a value of 00 through FF.

[BLINK- **][** BRIGHT- **][** *fgnd-color* **][** *bgnd-color* **]**

Names specific colors you want to use for the screen foreground and background. You use this option only if you are using Windows, and usually only if you use a color monitor. The following table lists the colors you can use for *fgnd-color* and *bgnd-color*.

Table 15: Windows colors

Color	Abbreviation
Black	Bla, Blk
Blue	Blu
Green	Gre, Grn
Cyan	C
Red	Red
Magenta	Ma
Brown	Bro, Brn
Gray	Gra, Gry
Dark-Gray	D-Gra

Color	Abbreviation
Light-Blue	Lt-Blu
Light-Green	Lt-Gre
Light-Cyan	Lt-C
Light-Red	Lt-Red
Light-Magenta	Lt-Ma
Light-Brown	Lt-Bro
Yellow	Y
White	W

If *fgnd-color* is omitted, then the system uses the foreground corresponding to NORMAL. If *bgnd-color* is omitted, then the system uses the background corresponding to NORMAL. If NORMAL, INPUT, or MESSAGES is specified for *fgnd-color* or *bgnd-color*, then the system uses the foreground or background color of the specified standard color.

[BLINK- **][** RVV- **][** UNDERLINE- **][** BRIGHT- **][** *fgnd-color* **]**

Names specific attributes you want to use for the screen display. Use this option only if you are using Windows, and usually only if you use a monochrome monitor. Normally, you would never specify *fgnd-color*.

VALUE (*expression*)

An expression with a value that results in one of the options in the COLOR phrase.

Example

The following procedure displays a random number of asterisks, in a random color, column, and row in 10 different occurrences. The COLOR statement displays the asterisks in one of the three colors stored in the elements of the hilite array. The COLOR phrase in this example is VALUE (hilite[RANDOM(1,3)]. The DISPLAY statement uses the color determined in the COLOR statement to display a random number of asterisks.

r-colphr.p

```
DEFINE VARIABLE hilite AS CHARACTER NO-UNDO EXTENT 3.
DEFINE VARIABLE loop   AS INTEGER   NO-UNDO.

hilite[1] = "NORMAL".
hilite[2] = "INPUT".  /* attribute to highlight */
hilite[3] = "MESSAGES".

REPEAT WHILE loop <= 10:
  FORM bar AS CHARACTER WITH ROW(RANDOM(3,17))
    COLUMN(RANDOM(5,50)) NO-BOX NO-LABELS
    FRAME bursts.
  COLOR DISPLAY VALUE(hilite[RANDOM(1,3)]) bar
    WITH FRAME bursts.
  DISPLAY FILL("*",RANDOM(1,8)) @ bar WITH FRAME bursts.
  PAUSE 1 NO-MESSAGE.
  HIDE FRAME bursts NO-PAUSE.
  loop = loop + 1.
END.
```

Notes

- For an application to use this COLOR phrase, it must use the default color table in the installed environment.
- The system ignores the color phrase entry for overlay frames on spacetaking terminals.
- For more information on the `protermcap` file, see *Manage ABL Applications*.

See also

COLOR statement

COLOR statement

Indicates the video attribute or color for normal display or for data entry.

Note: Does not apply to SpeedScript programming.

Syntax

```
COLOR [ DISPLAY ] color-phrase [ PROMPT color-phrase ]
      { field ... } { [ frame-phrase ] }
```

```
COLOR PROMPT color-phrase
      { field ... } { [ frame-phrase ] }
```

DISPLAY

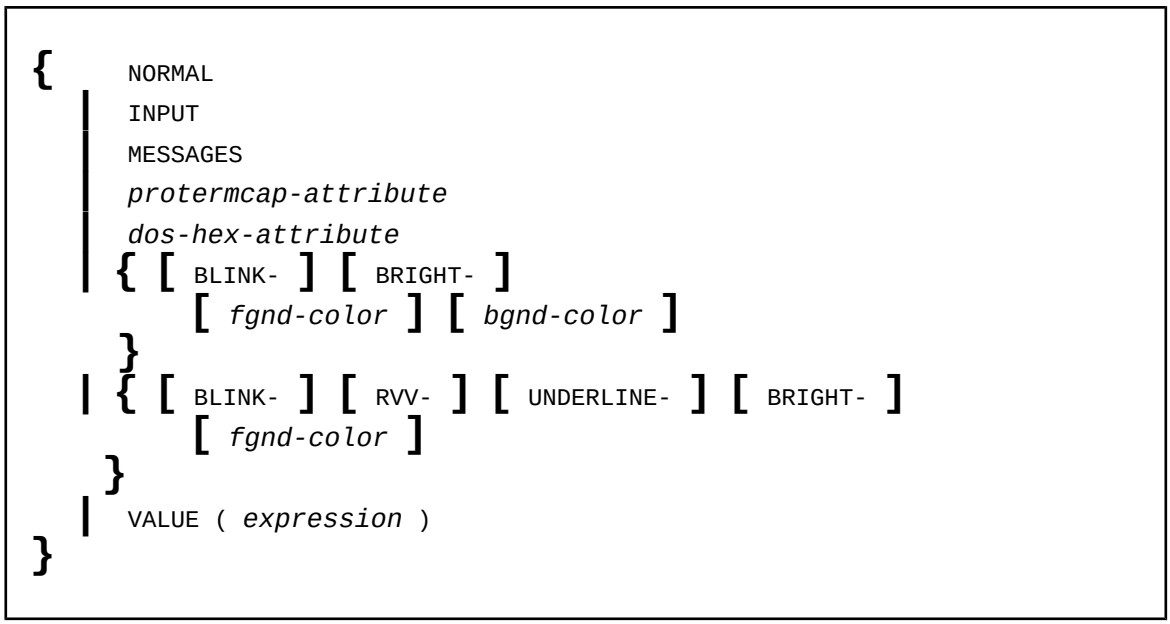
Indicates that you want to use a specific color when the system displays a field.

PROMPT

Indicates that you want to use a specific color when the system prompts a user for input by an INSERT, PROMPT-FOR, SET, or UPDATE statement.

color-phrase

Specifies a video attribute or color. Following is the syntax for *color-phrase*:



For more information on *color-phrase*, see the COLOR phrase reference entry. ABL ignores the *color-phrase* entry for overlay frames on spacetaking terminals.

field

The name of the field or fields for which you want to override the default colors.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information see the Frame phrase reference entry.

Example

This procedure highlights the item number and on-hand fields for items with an on-hand value less than 50. The variable hilite holds the video attribute (color) for highlighting. In this case, the system uses whatever attribute is used for the message area (such as reverse video, bright, or a color).

r-color.p

```
DEFINE VARIABLE hilite AS CHARACTER NO-UNDO INITIAL "messages".

/* Use standard messages attribute to highlight OnHand less than 50 */
FOR EACH Item NO-LOCK:
  DISPLAY Item.ItemNum Item.ItemName Item.OnHand.
  IF Item.OnHand < 50 THEN
    COLOR DISPLAY VALUE(hilite) Item.ItemNum Item.OnHand.
END.
```

Notes

- When the output destination is not the terminal, ABL disregards the COLOR statement.
- The COLOR statement does not automatically display a frame whose field's color attribute is changing.
- Use one of these statements to reset a field to the ABL default colors:

```
COLOR DISPLAY NORMAL PROMPT INPUT field
```

  Or:

```
COLOR DISPLAY NORMAL PROMPT INPUT field WITH FRAME frame
```

- Certain terminals, such as the WYSE 75, are non-spacetaking for some attributes and spacetaking for others.
- On UNIX, if you specify a color or video attribute that is not defined for the terminal, the AVM uses normal display instead.

See also

COLOR phrase, DISPLAY statement, Frame phrase

COMBO-BOX phrase

Describes a combo-box widget. A combo-box represents a field or variable, and consists of a field value and an associated drop-down list of possible values.

Note: Does not apply to SpeedScript programming.

Syntax

```
COMBO-BOX
  [ LIST-ITEMS item-list | LIST-ITEM-PAIRS item-pair-list ]
  [ INNER-LINES lines ] [ size-phrase ] [ SORT ]
  [ TOOLTIP tooltip ]
  [ SIMPLE | DROP-DOWN | DROP-DOWN-LIST ]
  [ MAX-CHARS characters ]
  [ AUTO-COMPLETION [ UNIQUE-MATCH ] ]
```

Use the following syntax to specify a combo-box widget for displaying values in a browse column:

```
COMBO-BOX
   [ LIST-ITEMS item-list | LIST-ITEM-PAIRS item-pair-list ]
   [ INNER-LINES lines ] [ SORT ]
   [ DROP-DOWN | DROP-DOWN-LIST ]
   [ MAX-CHARS characters ]
   [ AUTO-COMPLETION [ UNIQUE-MATCH ] ]
```

`LIST-ITEMS item-list`

Specifies that the items appear in the drop-down list. *item-list* represents a comma-separated list of valid values for the field or variable.

`LIST-ITEM-PAIRS item-pair-list`

Specifies a list of label-value pairs. Each pair represents the label and value of a field or variable. When the drop-down list appears, it displays each pair's label. Then, if the user selects a label, the AVM assigns the corresponding value to the field or variable. The syntax for *item-pair-list* is as follows:

```
label , value [ , label , value ] ...
```

label

A character string representing the label of the field or variable.

value

A value that the AVM assigns to the field or variable if the user selects the corresponding label.

`INNER-LINES lines`

Specifies the number of lines visible in the drop-down list for a DROP-DOWN or DROP-DOWN-LIST combo-box widget. The value for *lines* must be 3 or greater. If the number of lines you specify is less than the number of items in the drop-down list, the list is scrollable.

The INNER-LINES option in a SIMPLE combo-box definition is ignored.

size-phrase

Specifies the outside dimensions (width and height) of the combo-box widget and its drop-down list using the SIZE phrase. You must specify a SIZE phrase in the definition of a SIMPLE or DROP-DOWN combo-box widget. The syntax for the SIZE phrase is as follows:

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

For more information, see the SIZE phrase reference entry.

Note: The *height* value is ignored for DROP-DOWN and DROP-DOWN-LIST combo-box widgets. The height is always set to the height of a fill-in for the current font.

This option is invalid for combo-box browse columns. The AVM sets the ROW-HEIGHT-CHARS and ROW-HEIGHT-PIXELS attributes for a browse to the height of the combo-box, by default. You can override these settings after the browse is created.

`SORT`

Specifies that list items be sorted prior to display.

`TOOLTIP` *tooltip*

Allows you to define a help text message for a text field or text variable. The AVM automatically displays this text when the user pauses the mouse button over a text field or text variable for which a tooltip is defined.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the tooltip is removed. No tooltip is the default. The TOOLTIP option is supported in Windows only.

This option is invalid for combo-box browse columns.

`SIMPLE`

Specifies a combo-box widget with a read/write edit control and a list that is always visible. This option is supported in graphical interfaces only, and only in Windows. If you specify a SIMPLE combo-box widget in a character interface, the AVM treats it as a DROP-DOWN-LIST combo-box widget.

This option is invalid for combo-box browse columns.

`DROP-DOWN`

Specifies a combo-box widget with a read/write edit control and a drop-down list that appears when you click the drop-down button. This option is supported in graphical interfaces only, and only in Windows. If you specify a DROP-DOWN combo-box widget in a character interface, the AVM treats it as a DROP-DOWN-LIST combo-box widget.

For combo-box browse columns, a drop-down button appears only when an editable combo-box column has focus. You can use a DROP-DOWN combo-box only for a CHARACTER column. If the current value of the combo-box is not a valid value in the combo-box's item list, the AVM still displays the value in the combo-box's edit control.

`DROP-DOWN-LIST`

Specifies a combo-box widget with a read-only edit control and a drop-down list that appears when you click the drop-down button. This is the default.

For combo-box browse columns, a drop-down button appears only when an editable combo-box column has focus. You can use a DROP-DOWN-LIST combo-box for a CHARACTER, DECIMAL, INTEGER, INT64, DATE, or LOGICAL column. If the current value of the combo-box is not a valid value, the AVM does not display the value in the combo-box's edit control when the combo-box receives focus.

MAX-CHARS *characters*

The maximum number of characters the edit control can hold. The *characters* parameter must be a positive integer constant. If *characters* is zero or the Unknown value (?), MAX-CHARS is set to 255 characters by default.

Use MAX-CHARS with only SIMPLE and DROP-DOWN combo-boxes. It is ignored for DROP-DOWN-LIST combo-boxes. This option is supported in graphical interfaces only, and only in Windows.

AUTO-COMPLETION

Specifies that the edit control automatically complete keyboard input to the combo-box, based on a potential match, by searching through the items in the drop-down list. This option is supported in graphical interfaces only, and only in Windows.

UNIQUE-MATCH

Specifies that the edit control complete keyboard input to the combo-box based on a unique match. This option is supported in graphical interfaces only, and only in Windows.

Examples

The first example, `r-combo.p`, views a date field as a combo-box. When you run this procedure, you can choose a date value from the drop-down list. When you choose a new value, the VALUE-CHANGED trigger updates the value of out-string to an event associated with the new date value. The example initializes the drop-down list by building a comma-separated list of values and then assigning the string to the LIST-ITEMS attribute of the combo-box.

r-combo.p

```
DEFINE VARIABLE hist-date  AS DATE      NO-UNDO FORMAT "99/99/9999"
  VIEW-AS COMBO-BOX LIST-ITEMS 07/04/1776, 07/11/1969, 09/10/1993.
DEFINE VARIABLE hist-event AS CHARACTER NO-UNDO INITIAL
  "Declaration of Independence,Man walks on moon,Progress OpenEdge ships".
DEFINE VARIABLE out-string AS CHARACTER NO-UNDO FORMAT "x(36)".

DEFINE FRAME main-frame
  hist-date out-string
  WITH NO-LABELS TITLE "Historic Events".

ON VALUE-CHANGED OF hist-date DO:
  out-string = ENTRY(SELF:LOOKUP(SELF:SCREEN-VALUE), hist-event).
  DISPLAY out-string WITH FRAME main-frame.
END.

ENABLE hist-date WITH FRAME main-frame.
APPLY "VALUE-CHANGED" TO hist-date IN FRAME main-frame.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

The following example, `r-combo2.p`, builds a combo-box based on field values from a database. It defines triggers that allow you to change the value of the combo-box without displaying the drop-down list. They allow you to scroll through the values using the **CURSOR-DOWN** and **CURSOR-UP** keys or to jump to a specific value by typing its first letter.

r-combo2.p

```
DEFINE VARIABLE ix          AS INTEGER   NO-UNDO.
DEFINE VARIABLE rep         AS CHARACTER NO-UNDO LABEL "Rep"
```

```
  VIEW-AS COMBO-BOX.
DEFINE VARIABLE temp-string AS CHARACTER NO-UNDO.

FORM rep WITH FRAME main-frame SIDE-LABELS.

ON ANY-PRINTABLE OF rep DO:
  /* Find the first entry in the drop-down list that begins with the character
     typed. Set the SCREEN-VALUE of the combo box to that value. */
  seek-item:
  DO ix = 1 TO SELF:NUM-ITEMS:
    IF SELF:ENTRY(ix) BEGINS LAST-EVENT:FUNCTION THEN DO:
      SELF:SCREEN-VALUE = SELF:ENTRY(ix).
      LEAVE seek-item.
    END.
  END.
  IF ix > SELF:NUM-ITEMS THEN BELL.
END.

ON CURSOR-DOWN OF rep DO:
  /* Change the SCREEN-VALUE of the combo box to the next value from the
     drop-down list. */
  ix = SELF:LOOKUP(SELF:SCREEN-VALUE).
  IF ix < SELF:NUM-ITEMS THEN
    SELF:SCREEN-VALUE = SELF:ENTRY(ix + 1).
END.

ON CURSOR-UP OF rep DO:
  /* Change the SCREEN-VALUE of the combo box to the prev value from the
     drop-down list. */
  ix = SELF:LOOKUP(SELF:SCREEN-VALUE).
  IF ix > 1 THEN
    SELF:SCREEN-VALUE = SELF:ENTRY(ix - 1).
END.
temp-string = "".

FOR EACH Salesrep NO-LOCK:
  temp-string = IF temp-string = "" THEN SalesRep.SalesRep
    ELSE temp-string + "," + SalesRep.SalesRep.
END.
ASSIGN rep:LIST-ITEMS IN FRAME main-frame = temp-string.
ENABLE rep WITH FRAME main-frame.
```

Notes

- When the drop-down list appears, if it contains the value associated with the field or variable, that value is initially highlighted. Otherwise, no value in the drop-down list is initially highlighted.
- The LIST-ITEMS option of the COMBO-BOX phrase requires a list of items (possibly quoted, depending on the combo-box's data type), such as ("a", "b", "c"), whereas the LIST-ITEMS attribute of a combo-box requires a quoted list of items, such as ("a, b, c").

  Similarly, the LIST-ITEM-PAIRS option of the COMBO-BOX phrase requires a list of items (possibly quoted, depending on the combo-box's data type), such as ("a", 1, "b", 2, "c", 3), whereas the LIST-ITEM-PAIRS attribute of a combo-box requires a quoted list of items, such as ("a, 1, b, 2, c, 3").
- If you do not specify the LIST-ITEMS or LIST-ITEM-PAIRS option, the screen value of the variable or field becomes the null string (" "). To display or set values in the combo-box, you must first set the LIST-ITEMS or LIST-ITEM-PAIRS attribute to assign a drop-down list that specifies the available values.
- If you specify the SORT option for a COMBO-BOX, then any items you add with the ADD-FIRST, ADD-LAST, or INSERT methods are added in sorted order rather than the order you specify.
- Windows allows the user to transfer focus to the drop-down list by pressing **ALT** and one of the letters in the label. This is called a mnemonic.

- When you use the SIMPLE and DROP-DOWN options to define a character-field or character-variable combo-box widget, the FORMAT string for the field or variable is ignored.
- Items in a combo-box are case insensitive.

See also

Format phrase, SIZE phrase, VIEW-AS phrase

COMPARE function

The COMPARE function compares two strings and lets you:

- Perform a raw compare, if desired
- Use a particular collation
- Turn case sensitivity on and off

COMPARE returns a LOGICAL value representing the result of the logical expression, where the comparison rules are defined by the combination of the operator, the comparison strength, and the collation.

Syntax

```
COMPARE ( string1 , relational-operator , string2 ,
          strength [ , collation ] )
```

string1

A CHARACTER or LONGCHAR expression that evaluates to the first string to be compared.

relational-operator

A CHARACTER expression that evaluates to one of the relational operators, which are: LT (or <), LE (or <=), EQ (or =), GE (or >=), GT (or >), NE (or <>), "BEGINS", and "MATCHES".

string2

A CHARACTER or LONGCHAR expression that evaluates to the second string to be compared.

strength

A CHARACTER expression that evaluates to the ABL comparison strength or the International Components for Unicode (ICU) comparison strength to apply.

The ABL comparison strengths include:

- **RAW** — The AVM compares the two strings using the numeric values in the current code page.
- **CASE-SENSITIVE** — The AVM performs a case-sensitive comparison of the two strings using the numeric values in either the collation table specified in *collation*, or the collation table of the client. If you specify this strength with an ICU collation, the AVM applies the ICU TERTIARY strength.
- **CASE-INSENSITIVE** — The AVM performs a case-insensitive comparison of the two strings using the numeric values in either the collation table specified in *collation*, or the collation

table of the client. If you specify this strength with an ICU collation, the AVM applies the ICU SECONDARY strength.

- **CAPS** — The AVM converts any lowercase letters in the two strings to uppercase letters, based on the settings of the Internal Code Page (`-cpinternal`) and Case Table (`-cpcase`) startup parameters, and then performs a raw comparison of the resulting strings. When neither string contains a wildcard character, this option behaves the same as the MATCHES operator.

The ICU comparison strengths include:

- **PRIMARY** — The AVM compares the base characters in the two strings.
- **SECONDARY** — The AVM compares the base characters and any diacritical marks in the two strings.
- **TERTIARY** — The AVM performs a case-sensitive comparison of the base characters and diacritical marks in the two strings.
- **QUATERNARY** — The AVM performs a case-sensitive comparison of the base characters and any diacritical marks in the two strings, and distinguishes words with and without punctuation. ICU uses this strength to distinguish between Hiragana and Katakana when applied with the ICU-JA (Japanese) collation. Otherwise, it is the same as TERTIARY.

Note: Use ICU comparison strengths only with ICU collations.

`collation`

A CHARACTER expression that evaluates to the name of an ABL collation table or ICU collation. By default, the AVM uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (`-cpcoll`) startup parameter take precedence over a collation specified for any database the AVM accesses during the session, except when the AVM uses or modifies pre-existing indexes. If you do not specify a collation with the `-cpcoll` startup parameter, the AVM uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, the AVM uses the collation rules with the default name "`basic`" (which might or might not exist in the `convmap.cp` file).

If *`strength`* is not RAW or CAPS, the collation must be either an ABL collation table in the `convmap.cp` file or an ICU collation, and must be a valid collation table for the code page corresponding to the `-cpinternal` startup parameter.

Notes

- If either or both strings evaluate to the Unknown value (?), COMPARE returns the value indicated in the following table.

Table 16: Relational operators and the Unknown value (?)

Relational operator	Only one string evaluates to ?	Both strings evaluate to ?
LT (or <)	FALSE	FALSE
LE (or <=)	FALSE	TRUE
EQ (or =)	FALSE	TRUE
GE (or >=)	FALSE	TRUE
GT (or >)	FALSE	FALSE

Relational operator	Only one string evaluates to ?	Both strings evaluate to ?
NE (or <>)	TRUE	FALSE
BEGINS	FALSE	TRUE
MATCHES	FALSE	TRUE

- COMPARE returns the Unknown value (?) if one of the following occurs:
  - *relational-operator* does not evaluate to a valid value.
  - *strength* does not evaluate to a valid value.
  - *collation* does not evaluate to a collation table residing in the `convmap.cp` file.
  - *collation* evaluates to a collation table that is not defined for the code page corresponding to the `-cpinternal` startup parameter.
- LONGCHAR variable values are converted to `-cpinternal` for comparison and must convert without error, or the AVM returns an error.
- With BEGINS, the language-sensitive rules are used only when *strength* is not RAW or CAPS.
- With MATCHES, CASE-SENSITIVE is treated as RAW, CASE-INSENSITIVE is treated as CAPS, and the collation is never used.

COMPILE statement

Compiles a procedure file or a class definition file. A compilation can last for a session, or you can save it permanently for use in later sessions (as an r-code file, which has a `.r` extension).

When you compile a class definition file, ABL compiles the class definition file identified in the COMPILE statement and all class files in its inherited class hierarchy, by default. You can direct ABL to compile only those class definition files in the class hierarchy that are not found in the cache, and cache any classes or interfaces it compiles during the session, by setting the MULTI-COMPILE attribute to TRUE.

Note: When you change the definition of a class, Progress Software Corporation recommends that you recompile all classes that inherit the modified class. This recommendation does not apply to method logic changes within a class.

After you compile a procedure file, you use the RUN statement to create an instance of the procedure, and you use a handle to access the procedure and its context. After you compile a class definition file, you use the NEW function (classes) to create an instance of the class, and you use an object reference to access the class-based object, as well as its data members, properties, and methods.

For more information about compiling procedure files, see *Develop ABL Applications*. For more information about compiling class definition files, see *Develop Object-oriented ABL Applications*.

Syntax



procedure-pathname | VALUE (*expression*)

Specifies the name and location of a procedure file you want to compile, where *procedure-pathname* is the literal procedure pathname and *expression* is a character expression that evaluates to the procedure pathname. This pathname can be a full (absolute) pathname or it can be a pathname (or procedure filename only) relative to PROPATH. The specified procedure filename must include the extension (`.p` or `.w`) whether you specify it alone or as part of a path. On UNIX, filenames are case sensitive, so you must enter them exactly as they are stored. In Windows, the pathname cannot contain characters outside of the non-Unicode code page. See *Internationalize ABL Applications* for more information about Unicode and code pages.

class-pathname | VALUE (*expression*)

Specifies the name and location of a class definition file you want to compile. This can be a literal full (absolute) pathname or a literal pathname relative to PROPATH. If it is a relative pathname, the class or interface type name defined in the file must match the pattern of this relative pathname. If the type name is not defined with a package, the relative pathname must specify only the class filename. In all cases, the class filename must include the `.cls` extension. On UNIX, the *class-pathname* and corresponding class or interface type name are case sensitive, so you must enter them exactly as they are stored. In Windows, the he *class-pathname* and corresponding class or interface type name cannot contain characters outside of the non-Unicode code page. See *Internationalize ABL Applications* for more information about Unicode and code pages. For more information on packages and class or interface type names, see the Type-name syntax reference entry.

OPTIONS *options-list* | OPTIONS-FILE { *options-file* | VALUE (*expression*) }

Enforces one or more rules during compilation. In the case of OPTIONS, the rules are specified by a character expression that evaluates to a comma-separated list of options. In the case of OPTIONS-FILE, the rules are specified in the same way, but supplied in a designated file.

The possible options are described in the following table:

Option	Description
require-full-names [:Error | :Warning]	All table and field names must appear as they are in the schema. The compiler's ability to implicitly resolve abbreviated names in tables is disabled. Optionally set severity level to `Warning` or `Error`.
require-field-qualifiers [:Error | :Warning]	All buffer references (including database tables, temp-tables, and buffers) must be fully qualified. The compiler's ability to implicitly resolve the buffer to which a field reference refers is disabled. Optionally set severity level to `Warning` or `Error`.
require-full-keywords [:Error | :Warning]	All language keywords must be fully spelled out. Optionally set severity level to `Warning` or `Error`.
require-return-values [:Error | :Warning]	In user-defined functions, non-VOID methods, and property getters, all logical code paths must have RETURN *value* statements. For more information, see Require-return-values compiler option. Optionally set severity level to `Warning` or `Error`.

Note that the severity level specifies what happens when there is a failure:

- `Error` generates a message for each failure, and prevents the generation of r-code.
- `Warning` generates a message for each failure, but allows the generation of r-code, provided that the only failures during compilation are related to the option. (Other types of failure can prevent r-code generation.)
- If no severity level is specified, the default is `Warning`.

For example, the following COMPILE statement enforces full table and field names, and fully qualified references with different severity levels.

```
COMPILE myproc.p OPTIONS "require-field-qualifiers:Error,require-full-names".
```

The code below shows access of a field with a fully qualified reference, followed by unqualified and abbreviated examples of the same statement. Only the first statement compiles without error if the code is compiled using the statement above.

```
/* Accessing the field "tcfield" in the table "tbhelper" */

tbhelper.tcfield = "doUpdate".

/* The following does not compile with "require-field-qualifiers:Error"
in effect. */

tcfield = "doUpdate".

/* The following compiles with "require-full-names" with default severity level
in effect but warning messages are generated. */

tbh.tcf = "doUpdate".
```

If *options-list* or the contents of the options-file resolve to the empty string ("") or white space only, then no options are applied. If *options-list* or the contents of the options-file resolve to the Unknown value (?), then the compiler ignores the OPTIONS or OPTIONS-FILE phrase. If an option listed is not valid, the compile statement fails with an error. When compiling a class hierarchy, the options set in the COMPILE statement are applied to all files compiled for the hierarchy.

You can also set these options using the OPTIONS attribute of the COMPILER system handle, which can be initialized with the Compiler Options (`-compileroptionsfile`) startup parameter. (See *Startup Command and Parameter Reference* for more information.) Options set using the OPTIONS or OPTIONS-FILE in the COMPILE statement override any options set in the OPTIONS attribute of the COMPILER system handle (unless *options-list* or the contents of the options-file resolve to the Unknown value (?), as described above).

Note: OPTION-FILE supports the use of the hashtag (#) to include comments in the options file. See the Compiler Options (`-compileroptionsfile`) startup parameter in *Startup Command and Parameter Reference* for an example.

SAVE **[** = *logical-expression* **][** INTO **{** *directory* **|** VALUE (*expression*) **}]**

Produces a file that contains the r-code for the procedure or class you are compiling.

If you specify a *logical-expression*, its value determines whether the STREAM-IO option is activated. If the *logical-expression* is evaluated to the Unknown value (?), a run-time error occurs.

When you compile a class definition file with the SAVE option, ABL produces an r-code file for the class definition file and all class files in its inherited class hierarchy. For example, if you compile a class definition file that has two classes in its inherited class hierarchy, ABL compiles three files and produces three r-code files.

These r-code files are saved across ABL sessions. If you do not use the SAVE phrase, the COMPILE statement produces r-code for the source procedure or class, but the r-code is not saved across ABL sessions. This r-code is a session-compile version of the procedure or class.

If you specify a *logical-expression*, its value determines whether the SAVE option is activated. If the *logical-expression* is evaluated to the Unknown value (?), a run-time error occurs.

The COMPILE SAVE statement produces r-code files with the name *procedure-pathname*.r or *class-pathname*.r, where *procedure-pathname* is the pathname of a procedure source file without the filename extension, and *class-pathname* is the pathname of a class source file without the filename extension. ABL ignores the filename extension of a procedure or class definition file and always creates r-code files that use the same filename with a .r extension. For example, if you supply a filename of test, test.p, or test.cls, COMPILE SAVE produces an r-code file with the name test.r. If you specify a filename of test.bp, COMPILE SAVE still produces an r-code file with the name test.r.

Caution: Where both procedure and class definition files compile to a .r file, be sure to use distinct filenames. If you have a procedure file and a class definition file with the same name, and you compile them both with COMPILE SAVE, the first .r file is overwritten by the second .r file.

By default, the r-code file is stored in the same directory as the source file. The r-code files for inherited (super) class definition files are also stored in the same directory as their respective source files.

If you use the SAVE INTO phrase, r-code files produced by a compilation can be saved in a different directory. See the Examples section and the Notes section of this reference entry for more information.

A newly created r-code file replaces any existing r-code file of the same name.

LISTING { *listfile* | VALUE (*expression*) }

Produces a compilation listing that includes:

- The name of the file containing the procedure or class you compile
- The date and time at the start of the compilation
- The number of each line in the procedure or class file
- The block number where each statement belongs
- The complete text of all include files (except encrypted include files) and the names of any subprocedures and user-defined functions

The *listfile* or VALUE (*expression*) identifies the name of the file in which you want to store the Compiler listing. If *expression* evaluates to the Unknown value (?), then ABL ignores the LISTING option. In Windows, the filename cannot contain characters outside of the non-Unicode code page.

APPEND [= *logical-expression*]

Appends the current listing to the contents of the listing file. If you do not use the APPEND option, ABL creates a new listing file, replacing any file of the same name.

If you specify a *logical-expression*, its value determines whether the APPEND option is activated. If the *logical-expression* is evaluated to the Unknown value (?), a run-time error occurs.

PAGE-SIZE *integer-expression*

Identifies the number of lines to a page in the listing file. The default page size is 55 and *integer-expression* must be between 10 and 127, inclusive.

PAGE-WIDTH *integer-expression*

Identifies the number of page columns in the listing file. The default page width is 80, and *integer-expression* must be between 80 and 255, inclusive. Add at least 12 spaces to the page width when you type the file. This allows you to list information that precedes each line of code, ensuring that the file appears in the listing output exactly as you typed it.

XCODE *expression*

Decrypts the source code in *procedure-pathname* or *class-pathname,* and any encrypted include files, using the decryption key *expression.* The decryption key is also used with any class or interface in the hierarchy or any OOABL reference in a procedure or class that is not in the class hierarchy.

When the COMPILE statement detects that a source file is encrypted, it performs the following checks:

- If the XCODE option is provided, then COMPILE uses the key specified by *expression*. If the key does not match the source file key, then the compilation fails with an error message. Use this option only when the encryption key is not the built-in (default) key or the encryption key for the session.
- If XCODE is not present, the COMPILE statement looks for a session-level encryption key in the XCODE-SESSION-KEY attribute of the SECURITY-POLICY handle. If COMPILE finds a session key and it does not match the source file key, then the compilation fails with an error message.
- If XCODE is not present and there is no session key, the COMPILE statement uses the default key. If the default key does not match the source file key, then the compilation fails with an error message. If a session key is in effect and you want a COMPILE statement to use the default key, then you must unset the XCODE-SESSION-KEY attribute by setting it to the Unknown value (?) before executing the COMPILE statement.
- The XCODE utility does not perform code page conversions and does not use the `-cpinternal` parameter when encrypting files. Therefore, the source code and key use the default code page of the operating system where you run the XCODE utility. If a different code page is in effect where XCODE-SESSION-KEY is set, then code page conversions may prevent the attribute key from matching the XCODE utility key and the compile fails. To prevent this case, use only US-ASCII characters, which are found in all code pages below code point 128.

Include files that are not encrypted are included and compiled in the standard manner.

Having the decryption key does not allow you to examine a decrypted version of the source code.

Note: You cannot use XCODE with the XREF, XREF-XML, STRING-XREF, or LISTING options together. Also, if the DEBUG-LIST option is used with an encrypted source file, the resulting debug file only contains a notice that the source file is encrypted.

XREF { *xreffile* | VALUE (*expression*) } [APPEND [= *logical-expression*]]

Returns reference information on ABL elements, cross-references between procedures and ABL objects, and cross-references between class or interface definition files and ABL objects. See Table 17: XREF types and XREF information for more information on the type of reference information returned.

Information can be written to the file *xreffile* or VALUE (*expression*). If *expression* returns the Unknown value (?), then ABL ignores the XREF option.

In addition, you can use the APPEND option to add information to an existing file. When you use APPEND, the first line for a procedure contains information for the COMPILE type. This allows you to easily find where the information for each compilation begins. If you specify a *logical-expression*, its value determines whether the APPEND option is activated. If *logical-expression* is evaluated to the Unknown value (?), a run-time error occurs.

XREF generates one unformatted, white space-separated line in *xreffile* for each referenced element. Each line has the following format:

```
source-namefile-nameline-numberxref-typexref-information
```

The *source-name* is the name of the procedure or class file you compile with the COMPILE XREF statement. The *file-name* is the name of the file containing the referenced element. For example, *file-name* could be an include file. Otherwise, *file-name* is the same as *source-name*. The *line-number* is the line number of the statement in *file-name* that contains the referenced element. The *xref-type* is the type of reference in the code and *xref-information* contains details about the reference.

The possible values for *xref-type* and *xref-information* appear in the following table.

Table 17: XREF types and XREF information

XREF type	XREF information
ACCESS	{ [DATA-MEMBER] [*database.*] *tablefield* [WORKFILE | TEMPTABLE] } | { SHARED *variable* } | { PUBLIC-DATA-MEMBER *class-name*:*data-member-name* } | { PACKAGE-PROTECTED-DATA-MEMBER *class-name*:*data-member-name* } | { PACKAGE-PRIVATE-DATA-MEMBER *class-name*:*data-member-name* } | { INHERITED-DATA-MEMBER *class-name*:*data-member-name* } | { PUBLIC-PROPERTY *class-name*:*property-name* } | { PACKAGE-PROTECTED-PROPERTY *class-name*:*property-name* } | { PACKAGE-PRIVATE-PROPERTY *class-name*:*property-name* } | { INHERITED-PROPERTY *class-name*:*property-name* } | { *sequence-name* SEQUENCE }
ANNOTATION	*string*
BLOCK-LEVEL	ON ERROR UNDO, THROW[9] [(-undothrow)[10]]
CAST	[FROM *source-class-name*] TO *target-class-name*
CLASS	*class-name*, [INHERITS *inherited-class-name* [(*inherited-class-name* . . .)]], [IMPLEMENTS *interface-name* [*interface-name*] . . .], [USE-WIDGET-POOL], [FINAL], [ABSTRACT], [SERIALIZABLE]
COMPILE	

XREF type	XREF information
	procedure | *class-file*
CONSTRUCTOR	{ { PUBLIC | PACKAGE-PROTECTED | PROTECTED | PACKAGE-PRIVATE } | { , STATIC } },,,*constructor-name*, *void*, [*parameter1* [, *parameter2*] . . .]
CPINTERNAL	*code-page-name-that-ABL-uses-in-memory*
CPSTREAM	*code-page-name-that-ABL-uses-for-stream-I/O*
CREATE	{ [DATA-MEMBER | INHERITED-DATA-MEMBER] *class-name:table-name* } | [*database.*] *table* [WORKFILE | TEMPTABLE] }
DATA-MEMBER	{ PUBLIC | PACKAGE-PROTECTED | PROTECTED | PACKAGE-PRIVATE }, [STATIC], [TEMPTABLE | BUFFER | QUERY | DATASET | DATASOURCE], *data-member*
DATASET	*dataset-name*, { PROTECTED }, [REFERENCE-ONLY], [NAMESPACE-URI *namespace*], [NAMESPACE-PREFIX *prefix*], *buffer-name1* [[*buffer-name2*] . . .], [DATALINKS]
DELETE	{ [DATA-MEMBER | INHERITED-DATA-MEMBER] *class-name*:*table* } | [*database.*] *table* [WORKFILE | TEMPTABLE]
DELETE-INSTANCE	*class-name*

[9] The compiler adds a line in the XREF output whenever it encounters a ROUTINE-LEVEL ON ERROR UNDO, THROW or BLOCK-LEVEL ON ERROR UNDO, THROW statement in source code. If both are present, you'll get two XREF output lines.

[10] (-undothrow) in the XREF output indicates that either block-level or routine-level error handling was implemented by the -undothrow startup parameter.

XREF type	XREF information
DESTRUCTOR	PUBLIC,,,,*destructor-name*,*void*,
DLL-ENTRY	*procedure-name*,, [*parameter1* [, *parameter2*]...]
EXTERN	*function-name*,*return-type*, [*parameter1* [, *parameter2*]...]
EVENT	{PUBLIC | PACKAGE-PROTECTED | PROTECTED | PACKAGE-PRIVATE}, [STATIC], [OVERRIDE],,[ABSTRACT], *event-name*, [*delegate-type-name*], *return-type*, [*parameter1* [, *parameter2*]...]
FOR EACH: JOIN BY SQLDB	Not applicable (Data Servers only)
FUNCTION	*function-name*,*return-type*, [*parameter1* [, *parameter2*]...]
GLOBAL-VARIABLE	*global-variable*
INCLUDE	*include-file-name*

XREF type	XREF information
INTERFACE	*interface-type-name*, [INHERITS *interface-type-name* [(*interface-type-name*...)]],,,
INVOKE	*class-name*:*method-name* [, *invocation-parameter1* [, *invocation-parameter2*]...]
METHOD	{PUBLIC | PACKAGE-PROTECTED | PROTECTED | PACKAGE-PRIVATE}, [STATIC], [OVERRIDE], [FINAL], [ABSTRACT], *method-name*, *return-type*, [*parameter1* [, *parameter2*]...]
NEW	*class-name* [, *invocation-parameter1* [, *invocation-parameter2*]...]
NEW-SHR-DATASET	*dataset-name*, { PROTECTED }, [REFERENCE-ONLY], [NAMESPACE-URI *namespace*], [NAMESPACE-PREFIX *prefix*], *buffer-name1*[[*buffer-name2*]...], [DATALINKS]
NEW-SHR-FRAME	*new-shared-frame*
NEW-SHR-TEMPTABLE	*temptable-name*
NEW-SHR-VARIABLE	*new-shared-variable*
NEW-SHR-WORKFILE	*new-shared-workfile* [LIKE [*database*.]*table*]
PROCEDURE[11]	*procedure-name*,, [*parameter1* [, *parameter2*]...]

XREF type	XREF information
PRIVATE-FUNCTION	*function-name, return-type,* **[***parameter1***[** *, parameter2* **] . . .]**
PRIVATE-PROCEDURE [11]	*procedure-name,,* **[***parameter1* **[** *, parameter2***] . . .]**
PROPERTY	**{**PUBLIC **|** PACKAGE-PROTECTED **|** PROTECTED **|** PACKAGE-PRIVATE**}** **[** STATIC **]**, **[** OVERRIDE **]**,,**[** ABSTRACT **]**, *property-name*
PUBLISH[12]	**[***class-name*: **]***event-name***|** (*exp*)
REFERENCE	**{[** *database*. **]***table* **[** *field* **]** **[** WORKFILE **]}|{** SHARED *variable* **|** **{[** DATA-MEMBER **|** INHERITED-DATA-MEMBER **]** *class-name*:*table***[***field* **]}**
ROUTINE-LEVEL	ON ERROR UNDO, THROW[9] **[** (-undothrow)[10] **]**
RUN[11]	*procedure-name* **|***value*(*exp*) **[** PERSISTENT **|** REMOTE **|** SINGLE-RUN **|** SINGLETON **|** SUPER **|** STORED-PROC **]**
SEARCH[13]	**{[** *database*. **]***table***|** **{[** DATA-MEMBER **|** INHERITED-DATA-MEMBER **]** *class-name*:*table* **}}{** WORKFILE **|** **{{***index***|** RECID **}[** TEMPTABLE **]** **[** WHOLE-INDEX | TABLE-SCAN **]}}**
SHR-DATASET	*dataset-name,* **{** PROTECTED **}**, **[** REFERENCE-ONLY **]**, **[** NAMESPACE-URI *namespace***]**, **[** NAMESPACE-PREFIX *prefix* **]**,

XREF type	XREF information
	buffer-name1 **[[** *buffer-name2* **] . . .],** **[** DATALINKS **]**
SHR-FRAME	*shared-frame*
SHR-TEMPTABLE	*temptable-name*

[11] Internal procedures do not appear in XREF output when called from the procedure in which they are defined. Internal procedures appear in XREF output only when called from a different procedure.

[12] PUBLISH FROM does not appear in XREF output.

[13] WHOLE-INDEX means that the selection criteria specified to search the table does not offer opportunities to use indexes that allow optimized key references (bracketed high and low values). Instead, the AVM must search the entire table using available indexes (often only the primary index) to satisfy the query, hence a WHOLE-INDEX search. Thus, depending on the query, you might be able to optimize the search by adding indexes. If TABLE-SCAN is specified in the search criteria, XREF replaces WHOLE-INDEX with TABLE-SCAN. See also Notes.

XREF type	XREF information
SHR-WORKFILE	*shared-workfile* **[** LIKE **[** *database.* **]***table* **]**
SORT-ACCESS	**{[** *database.* **]** *tablefield* **[** WORKFILE **|** TEMPTABLE **]}**
SORT-BY-EXP	**{** FOR EACH **|** OPEN QUERY **}***table* BY *expression*
STRING	*char-string max-length justification* *translatable* **[** FORMAT **]**
SUBSCRIBE[14]	**[***class-name*: **]***event-name* **|** (*exp*) **[** , **[***class-name*: **]***handler-name* **[** , *parameter1* **[** , *parameter2***]...]]**
UNSUBSCRIBE[14]	**[***class-name*: **]***event-name* **|** (*exp*) **|** ALL **[** , **[***class-name*: **]***handler-name* **[** , *parameter1* **[** , *parameter2***]...]]**
UPDATE	

XREF type	XREF information
	{ [DATA-MEMBER] [*database*.] *table field* [WORKFILE | TEMPTABLE] } | { SHARED *variable* } | { PUBLIC-DATA-MEMBER *class-name*:*data-member-name* } | { PACKAGE-PROTECTED-DATA-MEMBER *class-name*:*data-member-name* } | { PACKAGE-PRIVATE-DATA-MEMBER *class-name*:*data-member-name* } | { INHERITED-DATA-MEMBER *class-name*:*data-member-name* } | { PUBLIC-PROPERTY *class-name*:*property-name* } | { PACKAGE-PROTECTED-PROPERTY *class-name*:*property-name* } | { PACKAGE-PRIVATE-PROPERTY *class-name*:*property-name* } | { INHERITED-PROPERTY *class-name*:*property-name* } | { *sequence-name* SEQUENCE }

This is the syntax for *data-member-name*:

```
variable-name | { [ dataset-name ] table [ field ] }
```

This is the syntax for *invocation-parameter*:

```
{ INPUT | INPUT-OUTPUT | OUTPUT }
{ [ data-type [ EXTENT [ constant ] ] ] |
  TABLE temp-table-name |
  TABLE-HANDLE |
  DATASET dataset-name |
  DATASET-HANDLE |
  TABLE-REFERENCE |
  DATASET-REFERENCE |
  RUNTYPE }
```

[14] SUBSCRIBE PROCEDURE and UNSUBSCRIBE PROCEDURE, which subscribes or unsubscribes another procedure to an event, does not appear in XREF output.

The invocation parameters are listed for every constructor or method invocation that has arguments. If a method or constructor is overloaded, invocation parameters can be used to determine which version of the method or constructor is being invoked. If the compiler has not resolved the method call (that is, it has deferred resolution to run time), one or more of the invocation parameters may be identified as TABLE REFERENCE, DATASET REFERENCE, or RUNTYPE. This label indicates that the argument being passed matches the type of the corresponding parameter in more than one of the candidate overloads.

In the case of TABLE REFERENCE or DATASET REFERENCE, the parameter may be a static temp table or dataset or a handle to a temp table or dataset.

RUNTYPE means that the compiler cannot determine the type because the argument is, for example, BUFFER-FIELD(1):BUFFER-HANDLE.

EXTENT without a constant integer value following it may simply mean that the parameter is defined as indeterminate, or it may mean that the call has not been resolved and the candidate overloads have arrays of different extents at that position in the parameter list.

The following notes describe more usage information for Table 17: XREF types and XREF information:

- PUBLIC-DATA-MEMBER, PACKAGE-PROTECTED-DATA-MEMBER, and PACKAGE-PRIVATE-DATA-MEMBER indicate that a line of code in a client of a class references a class public data member through an object reference. For example: `localvar = MyInstance:PubMember`. Properties are treated similarly.
- In contrast to PUBLIC-DATA-MEMBER, INHERITED-DATA-MEMBER indicates that a line of code references a data member inherited by the class in which the reference appears. DATA-MEMBER in an UPDATE or ACCESS entry indicates that a line of code references a temp-table or related object that has been defined in the class in which the reference appears. Properties are treated similarly.
- There are quotes around the name of a class or interface if its package name includes a space.
- In the CLASS entry, INHERITS *inherited-class-name* indicates the immediate super class of the class, if any exists. If the super class inherits from one or more classes, the names of these classes appear following the name of the immediate super class of the compiled class. Each inherited class name is separated from the preceding one by a space.
- XREF output includes "STRING" reference type entries. In addition to the strings that are already logged (variable names, function and procedure names, and so on), the XREF output now includes "STRING" reference type entries for class names, inherited class names, implemented interface names, method names, property names, and data member names.
- If a class has a super class but does not explicitly execute the SUPER statement in its constructor, the ABL compiler adds an implicit SUPER invocation to the r-code. When this happens, there is an INVOKE entry generated for the implicit SUPER invocation. This entry indicates that the name of the method being invoked is *super-class-type-name*:*class-name* (the constructor name). Instead of a line number, the entry uses the label IMPLICIT.
- As is the case with PROCEDUREs and FUNCTIONs, the entry for a method, constructor, or destructor is made during compilation of the element's END statement. Therefore, a METHOD entry appears after the entries for items encountered within the method, and the line number given is the line number of its END statement.
- Note that *field* for REFERENCE is optional. It does not appear if the corresponding line of code is either VALIDATE *temp-table* or RELEASE *temp-table*.
- If a class has as a data member an ABL handle and code uses that handle to call a built-in ABL method or to set or get a built-in attribute, the XREF output includes just an ACCESS entry and the entry identifies only the name of the handle data member, not the method or attribute involved. (This is similar to how XREF handles SHARED variables that are handles.)

- The XREF entry for CAST includes the source type if it is possible to determine the source type at compile time. If not, it includes only the target type.

Note: You cannot use the XREF and XCODE options together. That is, you cannot create a cross-reference listing from code that is encrypted.

`XREF-XML { directory | filename | VALUE ( expression ) }`

Writes reference information to a formatted XML file. The standard XREF option writes the information to an unformatted text file. The XREF-XML option provides structured output that is formatted with white space for easier parsing by humans. More importantly, this option exposes the reference information in a format that developers can exploit with custom-built tools or visualize in a ProDataset or a .NET dataset.

Table 17: XREF types and XREF information and the accompanying documentation in the XREF option section describes the XREF type and XREF information returned by either the XREF or the XREF-XML options. When you use XREF-XML, the XREF type is assigned to the `Reference-type` attribute. However, XREF information can be assigned to a number of XML elements including the `Object-identifier` attribute.

For compiling a single procedure or class, you can provide a filename for the XML output file. However, this output file is overwritten each time the compiler needs to compile a linked class or procedure.

For compiling several procedures and classes in a single compile statement, supply a *`directory`* for the XREF-XML option. The XREF-XML option uses this directory and a standard naming convention to capture the cross-reference information from multiple procedures and classes in separate files. Contrast this with the APPEND mode used by the XREF option.

When a directory is supplied, the compiler takes the root name of the procedure or class being compiled and creates a cross-reference file with this name and a `.xref.xml` file extension (*`sourcefilename.xref.xml`*). It stores it in the directory path specified, creating any necessary subdirectories that do not exist. If the main directory specified does not exist, then an error is raised.

If the filename supplied to the compiler begins with a relative path, then the directory name supplied to the XREF-XML option includes that relative path. For example, suppose your Unix PROPATH is `/projectA/source` and you run this COMPILE statement:

```
COMPILE test/procedureA.p XREF-XML /projectA/xref
```

The COMPILE statement looks for the source file `/projectA/source/test/procedureA.p` and place the XREF-XML output in `/projectA/xref/test/procedureA.xref.xml` (if the source file successfully compiles).

If the source file uses a full path, then the XREF-XML option stores the output in the directory provided, ignoring the path of the source file.

You may want to check for possible filename collisions before using this option. For example, if you have `myCode.p` and `myCode.cls`, both use the `myCode.xref.xml` output file, destroying some of your cross-reference information.

When you use the VALUE option to provide a filename or directory path, if VALUE returns the empty string or the Unknown value (?), then the compiler ignores the XREF-XML option.

If class definition source files in a class hierarchy are in different directories, matching subdirectories are created for them under the provided XREF-XML directory.

The XML Schema used with XREF-XML output files is stored in the following location: `$DLC/properties/schemas/xrefd`*`xxxx`*`.xsd`. The *XXXX* portion of the file name indicates the version number of the file.

You can see an example on the difference between XREF and XREF-XML output in the Examples section at the end of the COMPILE statement reference.

Note: You cannot use the XREF-XML option with the XREF option or with the XCODE option at the same time.

```
STRING-XREF { sxreffile | VALUE ( expression ) }
[ APPEND [ = logical-expression ] ]
```

Writes cross-reference string information between procedures and ABL objects, or between class definition files and ABL objects, to the file *sxreffile* or VALUE (*expression*). If *expression* evaluates to the Unknown value (?), ABL ignores the STRING-XREF option.

```
String Xref Version x.ysource-filecode-page
```

The *x.y* is a major.minor version number, where a major version change implies a formatting change that is not backward compatible with older versions of TranManII. The *source-file* is the name of the file from which the strings are extracted. The *code-page* is the code page with which the file was written.

The line for each string appears in the following format:

```
line-number object-name string max-length string-justification
statement-type detail-info
```

The *line-number* is the same as *line-number* in the standard XREF file. The *object-name* is the name of the object with which the string is associated. The *max-length* and *string-justification* come from the string attribute (either explicit or implicit) and reflect the attributes applied to the string as it is entered into the text segment.

The *statement-type* describes the type of statement in which the string appears. Only one statement type appears in a given string's output line. The values in the following table are possible:

Statement type values			
ASSIGN	DEF-SUB-MENU	INSERT	PUT-SCREEN
CASE	DISPLAY	MESSAGE	REPEAT
CREATE	DO	OPEN-QUERY	RUN
DEF-BROWSE	ENABLE	OTHER	SET
DEF-BUTTON	EXPORT	PAUSE	STATUS
DEF-FRAME	FOR	PROMPT-FOR	UPDATE

Statement type values			
DEF-IMAGE	FORM	PUT	VIEW-AS
DEF-MENU	IF	–	–

Note: Any statement type that is not included in the preceding list appears as OTHER.

The *detail-info* is one or more detail tags that specify more specifically where the string appears in the statement. The values in the following table are possible:

Detail tags			
ASSIGN	FORMAT	MESSAGE	TITLE
COL-LABEL	IMAGE-FILE	NON-ALPHA	VALUE
COMBO-BOX-ITEM	INPUT	PROMSGS	WHEN
CUR-LANG	INPUT-PARAM	PROPATH	WHERE
DEFAULT	LABEL	SEL-LIST-ITEM	WHILE
EXPR	LIST-ITEM	TERMCAP	–

Note: The NON-ALPHA tag indicates that a string consists entirely of blanks or digits. The FORMAT tag is followed by one of the following tags: CHAR, NUMERIC (includes decimal and integer), DATE, or BOOL. These tags indicate the type of format. When a string can appear in only one place in a statement, no detail tag appears.

The following table shows the valid combinations of statement types and detail tags.

Table 18: Valid statement type and detail tag combination

Statement type	Detail tags
ASSIGN	CUR-LANG, PROMSGS, PROPATH, TERMCAP
CASE	WHEN
CREATE	N/A
DEF-BROWSE	FORMAT, COL-LABEL
DE-FBUTTON	IMAGE-FILE, LABEL
DEF-FRAME	FORMAT, COL-LABEL, LABEL
DEF-IMAGE	IMAGE-FILE
DEF-MENU	TITLE, LABEL

Statement type	Detail tags
DEF-SUB-MENU	LABEL
DISPLAY	FORMAT, LABEL, COL-LABEL, WHEN, TITLE
DO	WHILE, WHERE, TITLE
ENABLE	LABEL, COL-LABEL, WHEN, TITLE
EXPORT	FORMAT
FOR	WHILE, WHERE, TITLE
FORM	FORMAT
IF	N/A
INSERT	TITLE
MESSAGE	TITLE, FORMAT
PAUSE	MESSAGE
PROMPT-FOR	WHEN, TITLE, FORMAT, LABEL, COL-LABEL
PUT	N/A
PUT-SCREEN	N/A
REPEAT	WHILE, TITLE, WHERE
RUN	INPUT-PARAM
SET	WHEN, ASSIGN, FORMAT, LABEL, COL-LABEL, TITLE
STATUS	DEFAULT, INPUT
UPDATE	WHEN, ASSIGN, FORMAT, LABEL, COL-LABEL, TITLE
VIEW-AS	SEL-LIST-ITEM, COMBO-BOX-ITEM

`STREAM-IO [ = logical-expression ]`

Specifies that all output from the compiled procedure or class is formatted for output to a file or printer. This means that all font specifications are ignored and all frames are treated as if they had the USE-TEXT option given. This produces a platform-independent output appropriate for printing.

`LANGUAGES ( { language-list | VALUE ( expression ) } )`

Identifies which language segments to include in the compiled r-code. The *language-list* is a colon-separated list of language names used to generate each text segment. If you specify VALUE

(*expression*), the expression must evaluate to a comma-separated list of language names. If *expression* evaluates to the Unknown value (?), then ABL ignores the LANGUAGES option.

Translated character strings for each specified language are read from the translation database and are stored in segments within the r-code. For example:

```
COMPILE myfile.p LANGUAGES
  (French-Canadian:French:English,Portuguese:Spanish,
   New-York:American:English).
```

If you use an expression to specify *language-list*, you must use the VALUE option. For example:

```
COMPILE myfile.p LANGUAGES (VALUE(char-var)).

/* char-var = "French-Canadian:French:English,Portuguese:Spanish,
  New-York:American:English" */
```

In this example, the compiler searches the translation database for French-Canadian translations. If a French-Canadian translation is not found, the compiler searches for a French translation. If a French translation is not found, the compiler searches for an English translation. If an English translation is not found, the compiler uses the strings from the source code.

This example generates four text segments: French-Canadian, Portuguese, New-York, and the unnamed (default) text segment. The first language name in each *language-list* argument designates the name of the text segment and specifies the first language that the compiler looks up in the translation database. As a result, it is possible to create a text segment whose name has no relationship to the languages it is composed of. For example, the following argument creates a text segment named BABEL:

```
LANGUAGES(BABEL:French:Spanish:Italian:German)
```

Provided there is no language named BABEL in the translation database, the strings in this text segment would be either French, Spanish, Italian, or German, depending on which strings have translations in which languages.

TEXT-SEG-GROW = *growth-factor*

Specifies the factor by which ABL increases the length of strings. When you develop an application that is going to be translated, it is important to allow for the growth of the text in your widgets. If you use the TEXT-SEG-GROW option, ABL increases the size of the text strings when it compiles your application.

ABL uses the following formula to determine the length of strings:

```
New-length =
 Actual-length * ( 1 + ( growth-factor/100 *
  ( table-value/100 ) ) )
```

Where:

- *New-length* is the new string length.
- *Actual-length* is the actual string length.

- *growth-factor* is the value specified with the TEXT-SEG-GROW option.
- *table-value* is the appropriate percentage from the following table:

String length	Expansion percentage
1-10 characters	200%
11-20 characters	100%
21-30 characters	80%
31-50 characters	60%
51-70 characters	40%
More than 70 characters	30%

For example, if you have a text string that is 25 characters and you specify a *growth-factor* of 50, ABL applies the formula as follows and defines the *New-length* as 35:

```
New-length = 25 * ( 1 + (80/100 * (50/100)) )
```

Note: TEXT-SEG-GROW is supported only when you also use the LANGUAGES option.

DEBUG-LIST { *debugfile* | VALUE (*expression*) }

Writes the debug listing to the file *debugfile* or VALUE (*expression*). If *expression* evaluates to the Unknown value (?), then ABL ignores the DEBUG-LIST option. The *debugfile* consists of a line-numbered listing of the procedure with the text of all preprocessor include files, names, and parameters inserted. In Windows, the filename cannot contain characters outside of the non-Unicode code page.

PREPROCESS { *preprocessfile* | VALUE (*expression*) }

Preprocesses the procedure or class definition file and writes the preprocessed source code to the file *preprocessfile* or VALUE (*expression*). If *expression* evaluates to the Unknown value (?), ABL ignores the PREPROCESS option. The *preprocessfile* is a text file that contains a final version of your source code after all include files have been inserted and all text substitutions have been performed. In Windows, the filename cannot contain characters outside of the non-Unicode code page.

V6FRAME [= *logical-expression*] [USE-REVVIDEO | USE-UNDERLINE]

The V6FRAME option is designed specifically to compile and run Progress Version 6 applications with Progress Version 7 or later in Windows. This option uses the V6FontNumber setting in the [Startup] section of the current environment (which might be the Registry or an initialization file) to calculate the height and width of a character unit and then set the layout grid used to compile frames for display in Progress Version 7 or later.

At run time, the FONT attribute for a frame compiled with the V6FRAME option is set to the font number specified with the V6FontNumber setting. The default setting for the V6FontNumber setting is 3.

By default, V6FRAME displays a border around a fill-in field. This means that your code requires more space on the screen than in Progress Version 6. You can override this behavior with one of the following options:

- USE-REVVIDEO displays no border around a fill-in field. When a fill-in is enabled for input, the color of the fill-in changes to the color specified with the INPUT setting in the [Colors] section in the current environment (which might be the registry or an initialization file). The IBEAM cursor signals that a fill-in field has input focus.
- USE-UNDERLINE displays no border around a fill-in widget. When a fill-in is enabled for input, the underline attribute of the font (V6FontNumber) for the fill-in is turned on. The color of a fill-in enabled for input does not change. The IBEAM cursor signals that a fill-in field has input focus.

The V6FRAME option also limits the vertical size of a frame title to one character unit based upon the layout grid. The text of the frame title is in the font specified with the V6FontNumber setting in the [Startup] section of the current environment (which might be the registry or an initialization file).

The V6FRAME option governs the appearance of screen output only. Use the STREAM-IO option to compile procedures that output to files and printers. If you specify the V6FRAME and STREAM-IO options in the same COMPILE statement, the STREAM-IO option overrides the V6FRAME option.

If you specify a *logical-expression*, its value determines whether the V6 compile option is activated. If the *logical-expression* is evaluated to the `Unknown value (?)`, a run-time error occurs.

For more information on the environment for an ABL session, see *Manage ABL Applications*.

`MIN-SIZE [ = logical-expression ]`

Minimizes the size of the generated r-code file by eliminating the Debugger Segment (which is used by the OpenEdge Debugger) and the signature descriptor data (which is used by the Open Client Proxy Generator).

If you specify a *logical-expression*, its value determines whether the MIN-SIZE option is activated (TRUE) or not (FALSE). If the *logical-expression* evaluates to the `Unknown value (?)`, a run-time error occurs. The default value is FALSE.

`NO-ERROR`

The NO-ERROR option is used to prevent the statement from raising `ERROR` and displaying error messages.

You can also check the `ERROR` and `WARNING` attributes of the `COMPILER` system handle to determine whether an error or warning occurred.

Examples

In this procedure, ABL compiles the `ord-ent` procedure, produces an r-code file, `ord-ent.r`, that can be used across ABL sessions, and saves the r-code file in the current directory:

r-cmple.p

```
COMPILE ord-ent SAVE.
```

Note: The sample procedures supplied with ABL do not include the `ord-ent` procedure.

You can save the r-code file in a different directory by using the SAVE INTO phrase. For example, to save an r-code file in `/usr/sources` on a UNIX system, enter this command:

```
COMPILE ord-ent SAVE INTO /usr/sources.
```

The following example shows the effect of include files on compilation listings:

r-incl.p

```
FOR EACH Customer NO-LOCK:
  {r-fcust.i}
  {r-dcust.i}
END.
```

Suppose you use the following COMPILE statement to compile the `r-incl.p` procedure:

r-comlis.p

```
COMPILE r-incl.p SAVE LISTING r-incl.lis XREF r-incl.xrf
  DEBUG-LIST r-incl.dbg.
```

This COMPILE statement produces four files: `r-incl.r`, `r-incl.lis`, `r-incl.xrf`, and `r-incl.dbg`.

The following procedures contain the contents of the `r-incl.lis`, `r-incl.xrf`, and `r-incl.dbg` files:

r-incl.lis

```
r-incl.p                       06/01/93 13:06:30   PROGRESS(R) Page 1

{} Line Blk
-- ---- ---
      1     /* r-incl.p */
      2
      3   1 FOR EACH Customer NO-LOCK:
      4   1   {r-fcust.i}
 1    1   1 /* r-fcust.i */
 1    2   1
 1    3   1 FORM Customer.CustNum Customer.Name LABEL "Customer Name"
 1    4   1   Customer.Phone FORMAT "999-999-9999".
      4   1
      5   1   {r-dcust.i}
 1    1   1 /* r-dcust.i */
 1    2   1
 1    3   1 DISPLAY Customer.CustNum Customer.Name Customer.Phone.
      5   1
      6     END.

^Lr-incl.p                     06/01/93 13:06:30   PROGRESS(R) Page 2

     File Name       Line Blk. Type Tran            Blk. Label
-------------------- ---- --------- ---- --------------------------------
r-incl.p                0 Procedure No
r-incl.p                3 For       No
    Buffers: Sports2020.Customer
    Frames:  Unnamed

^L
```

This sample output is not an exact copy of the `r-incl.lis` file.

There are three columns next to the procedure in the listing file:

1. **{}** — The level of the include file
2. **Line** — The line number in the file
3. **Blk** — The number of the block

The information follows each of the procedure blocks or function blocks:

- **Line** — The line number where the block starts
- **Blk. Type** — The type of block (Procedure, DO, FOR EACH, REPEAT)
- **Tran** — Whether the block is a transaction block
- **Blk. Label** — The label of the block
- **Buffers** — The name of the record buffer scoped to the block
- **Frames** — The name of the frame scoped to the block

This is the cross-reference file `r-incl.xrf`:

r-incl.xrf

```
r-incl.p r-incl.p 1 COMPILE r-incl.p
r-incl.p r-incl.p 3 STRING "Customer" 8 NONE UNTRANSLATABLE
r-incl.p r-incl.p 3 SEARCH Sports2020.Customer CustNum
r-incl.p r-incl.p 4 INCLUDE r-fcust.i
r-incl.p r-fcust.i 3 ACCESS Sports2020.Customer CustNum
r-incl.p r-fcust.i 3 ACCESS Sports2020.Customer Name
r-incl.p r-fcust.i 3 ACCESS Sports2020.Customer Phone
r-incl.p r-fcust.i 3 STRING ">>>>9" 5 NONE TRANSLATABLE  FORMAT
r-incl.p r-fcust.i 3 STRING "x(20)" 5 NONE TRANSLATABLE  FORMAT
r-incl.p r-fcust.i 3 STRING "999-999-9999" 12 NONE TRANSLATABLE  FORMAT
r-incl.p r-incl.p 5 INCLUDE r-dcust.i
r-incl.p r-dcust.i 3 ACCESS Sports2020.Customer CustNum
r-incl.p r-dcust.i 3 ACCESS Sports2020.Customer Name
r-incl.p r-dcust.i 3 ACCESS Sports2020.Customer Phone
r-incl.p r-incl.p 6 STRING "CustNum" 8 LEFT TRANSLATABLE
r-incl.p r-incl.p 6 STRING "Customer Name" 13 LEFT TRANSLATABLE
r-incl.p r-incl.p 6 STRING "Phone" 5 LEFT TRANSLATABLE
r-incl.p r-incl.p 6 STRING
    "-------- ---------------------- --------------" 46 LEFT TRANSLATABLE
r-incl.p r-incl.p 6 STRING "CustNum" 8 LEFT TRANSLATABLE
```

Each line in the `xref` file specifies the procedure, line number, access type, and access information. The first line in the `xref` file contains the COMPILE access type directive and the name of the procedure exactly as it appears in the COMPILE statement. See Table 17: XREF types and XREF information for a list of the values that follow a particular access type (for example, *`table`* and *`index`* after SEARCH).

If you modified `r-comlis.p` to use the XREF-XML option instead of XREF, your cross reference file would be named `r-comlis.xref.xml`. The structured formatting of XML would use many more lines to display the same information on one line of standard XREF output. Here is a small snippet of that file:

```
...
<Reference Reference-type="INCLUDE" Object-identifier=""c:\openedge\wrk\r-fcust.i"">
  <Source-guid>9+5Nn2plMK7bEdVPTNq+Gw</Source-guid>
  <File-num>1</File-num>
  <Ref-seq>6</Ref-seq>
  <Line-num>4</Line-num>
  <Object-context />
  <Access-mode />
  <Data-member-ref />
  <Temp-ref />
  <Detail />
  <Is-static>false</Is-static>
  <Is-abstract>false</Is-abstract>
</Reference>
<Reference Reference-type="INCLUDE" Object-identifier=""c:\openedge\wrk\r-dcust.i"">
  <Source-guid>9+5Nn2plMK7bEdVPTNq+Gw</Source-guid>
  <File-num>1</File-num>
  <Ref-seq>13</Ref-seq>
  <Line-num>5</Line-num>
  <Object-context />
  <Access-mode />
  <Data-member-ref />
  <Temp-ref />
  <Detail />
  <Is-static>false</Is-static>
  <Is-abstract>false</Is-abstract>
</Reference>
<Reference Reference-type="STRING" Object-identifier="Cust Num">
  <Source-guid>9+5Nn2plMK7bEdVPTNq+Gw</Source-guid>
  <File-num>1</File-num>
  <Ref-seq>17</Ref-seq>
  <Line-num>6</Line-num>
  <Object-context />
  <Access-mode />
  <Data-member-ref />
  <Temp-ref />
  <Detail />
  <Is-static>false</Is-static>
  <Is-abstract>false</Is-abstract>
  <String-ref>
    <Source-guid>9+5Nn2plMK7bEdVPTNq+Gw</Source-guid>
    <Ref-seq>17</Ref-seq>
    <Max-length>8</Max-length>
    <Justification>LEFT</Justification>
    <Translatable>true</Translatable>
  </String-ref>
</Reference>
...
```

This is the debug listing `r-incl.dbg`:

r-incl.dbg

```
 1   /* r-incl.p */
 2
 3   FOR EACH Customer NO-LOCK:
 4
 5   /* r-fcust.i */
 6
 7   FORM Customer.CustNum Customer.Name LABEL "Customer
 8         Name" Customer.Phone FORMAT "999-999-9999".
 9
10
11   /* r-dcust.i */
12
13  DISPLAY Customer.CustNum Customer.Name Customer.Phone
14
15   END.
```

For a class that implements an interface hierarchy or an interface that inherits an interface hierarchy, the whole of the hierarchy appears in the XREF and XREF-XML as a complete list. The following example illustrates the comparison of XREF output both with and without interface inheritance.

XREF output without interface inheritance

```
INTERFACE interface-name,,,,
```

XREF output with interface inheritance

```
INTERFACE interface-name,INHERITS acme.inventory.IWarehouse
(acme.inventory.IStoreRoom acme.inventory.IShelf),,,,
```

Because interfaces support multiple inheritance, the order of the interface hierarchy cannot be determined.

The following example is for the XREF-XML output both with and without interface inheritance.

XREF-XML output without interface inheritance

```
<Reference Reference-type="INTERFACE" Object-identifier="interface-name">
  <Source-guid>7mu7M0PmmZ0R3/Mtc60gbA</Source-guid>
  <File-num>1</File-num>
  <Ref-seq>4</Ref-seq>
  <Line-num>4</Line-num>
  <Object-context/>
  <Access-mode/>
  <Data-member-ref/>
  <Temp-ref/>
  <Detail/>
  <Is-static>false</Is-static>
  <Is-abstract>false</Is-abstract>
  <Interface-ref>
    <Source-guid>7mu7M0PmmZ0R3/Mtc60gbA</Source-guid>
    <Ref-seq>4</Ref-seq>
    <Inherited-list/>
  </Interface-ref>
</Reference>
```

XREF-XML output with interface inheritance

```
<Reference Reference-type="INTERFACE" Object-identifier="interface-name">
  <Source-guid>7mu7M0PmmZ0R3/Mtc60gbA</Source-guid>
  <File-num>1</File-num>
  <Ref-seq>4</Ref-seq>
  <Line-num>4</Line-num>
  <Object-context/>
  <Access-mode/>
  <Data-member-ref/>
  <Temp-ref/>
  <Detail/>
  <Is-static>false</Is-static>
  <Is-abstract>false</Is-abstract>
  <Interface-ref>
    <Source-guid>7mu7M0PmmZ0R3/Mtc60gbA</Source-guid>
    <Ref-seq>4</Ref-seq>
    <Inherited-list/>acme.inventory.IWarehouse(acme.inventory.IStoreRoom
    acme.inventory.IShelf)]</Inherited-list>
  </Interface-ref>
</Reference>
```

Notes

- When compiling class definition files, the following options apply to the class definition file identified in the COMPILE statement and all class files in its inherited class hierarchy: XCODE, STREAM-IO, LANGUAGES, V6FRAME, and MIN-SIZE.
- When compiling class definition files, the following options apply only to the class definition file identified in the COMPILE statement, and not to the class files in its inherited class hierarchy: PREPROCESS, LISTING, DEBUG-LIST, XREF, and STRING-XREF.
- If you want all record retrieval statements in a procedure to default to NO-LOCK, you must compile the procedure in an ABL session started with the No Lock (`-NL`) startup parameter. For more information on record locking, see *Develop ABL Applications*. For more information on the No Lock (`-NL`) startup parameter, see *Startup Command and Parameter Reference*.
- Two additional startup parameters are available that can affect the behavior of the `COMPILE` statement: Compile Warning List (`-cwl`) and Keyword Forget List (`-k`). The Compile Warning List option lets you specify a set of ABL statements that trigger compile-time warnings if they are found in the source code. The Keyword Forget List option lets you disable specified keywords, allowing you to compile code in which those keywords occur as ABL user-defined element names, such as table, field, variable, and procedure names. For more information about these startup parameters, see *Startup Command and Parameter Reference*.
- The value of the `PROPATH` environment variable defines the list of directories (path) to use when searching for a procedure.
- The ABL compiler does not search procedure libraries for include files referenced in a procedure.
- On UNIX, you define the `PROPATH` variable in a startup script or in your `.profile` file. In Windows, you can define your `PROPATH` in the Registry or in an initialization file. You can also define the `PROPATH` interactively at the operating system level.

  In addition to any directories you define for `PROPATH`, ABL searches the directory containing the ABL system software. If you do not define a value for `PROPATH`, ABL searches your working directory by default.
- To locate the source file that you name in the COMPILE SAVE statement, ABL searches the first directory in `PROPATH`. If the source file is there, ABL compiles the source file and creates an r-code file. On UNIX, this new r-code file replaces any existing r-code file. If errors occur during compilation, ABL does not produce an r-code file and leaves existing r-code files unchanged.

  If ABL cannot find the source file, it continues on to the next directory in `PROPATH`.

- Use the SAVE INTO phrase to store a compiled r-code file in a different directory from its corresponding source file.

  If you specify a relative pathname for the source file, that pathname is appended to the SAVE INTO path. For example (using UNIX pathnames):

```
PROPATH="/pro1/source".
COMPILE test/proc1.p SAVE INTO /pro1/obj.
```

  In the example, ABL saves the source file `/pro1/source/test/proc1.p` as `/pro1/obj/test/proc1.r`.

  If the source file is a full pathname, ABL stores the r-code file in the SAVE INTO directory; it drops its original directory path. For example:

```
COMPILE /pro1/obj/test/proc1.p SAVE INTO /usr/rcode.
```

  In the example, ABL saves the source file as `/usr/rcode/proc1.r`.

  When you use the SAVE INTO phrase to store compiled r-code files for one or more class definition files specified with a package, ABL creates a directory structure under the specified SAVE INTO directory that is consistent with the directory structure of the original source files relative to PROPATH (if the directory structure doesn't already exist). That is, ABL creates a subdirectory under the specified SAVE INTO directory to match the original source directory for each class definition file in the hierarchy.

  For example, if the source for two class definition files in a class hierarchy reside in two different directories, such as `dir1` and `dir2`, ABL creates two matching subdirectories named `dir1` and `dir2` under the specified SAVE INTO directory and stores the r-code files in their respective subdirectories.

  If the SAVE INTO pathname is null, ABL saves r-code files in the same directory as their source files.

- To locate a file with the COMPILE statement (without the SAVE phrase), ABL searches the first directory in PROPATH for a usable r-code file. A usable r-code file must meet these criteria:

  - It must have the correct format; it must have been produced by the COMPILE SAVE statement.
  - It must have been produced by the current version of the ABL compiler.
  - It must have the same cyclic redundancy check (CRC) value as any database tables it references. When creating an r-code file, ABL includes, as part of the r-code file, the CRC of the most recent change to the database schema that affects **this** procedure (for example, adding or deleting a field or index definition in a table that the procedure references).
  - On UNIX, it must have read access to the r-code file.

  If there is a usable r-code file, there is no reason to perform the compilation. You receive an error and the compilation stops unless you have specified the XREF, LISTING, PREPROCESS, or DEBUG-LIST option. If you specified one of these options, ABL continues with the compilation and produces the files specified and a session compile. If ABL does create a session compile version, the version is not used when you use the RUN statement. The RUN statement always uses an existing r-code file before using a session compile version of a procedure.

  If there is no usable r-code file, ABL searches the same directory in PROPATH for a source file. If the source file is there, ABL compiles it into the session compile file. If it is not there, ABL continues on to the next directory in PROPATH, searching for an r-code file, then for a source file.

- After you compile a procedure, the RUN statement does not recompile it. If you RUN a procedure multiple times within a session, changing the procedure between runs, you must manually recompile the procedure

each time. Otherwise, the procedure's last r-code, which persists for a session, is found and the procedure is not automatically recompiled.

- The size of the r-code might vary, depending on the window system on which it is compiled.
- Modifications to existing field definitions do not affect database table CRC or time-stamp values. Therefore, updating a table's existing field definitions does not invalidate r-code versions of procedures that reference the table. However, adding or deleting tables, fields, or indexes does affect database table CRC and time stamps. This invalidates r-code versions of procedures that reference the changed tables.
- When you use a reserved keyword to specify a language with the LANGUAGES option, you must use quotation marks (" ") around the *language-list*.
- The SORT-BY-EXP reference in the XREF is used to indicate a FOR EACH or OPEN QUERY statement which contains a BY clause which uses an expression.
- A WHOLE-INDEX search reported for a table occurs when an entire index is used to search the table. (That is, the bracket used by the query to search the table spans the entire index.) This can occur either when no selection criteria are specified to limit the range of index keys searched (that is, to bracket a subset of the index) or when there is no appropriate index available to optimize the selection criteria. For example, the following queries on `Customer` table of the `Sports2020` database both result in WHOLE-INDEX searches. The first query uses the `Name` index to search the entire table, returning every record in Name order. The second query uses the primary index to search the entire table because there is no index provided for the Balance field to limit the search.

```
FOR EACH Customer NO-LOCK USE-INDEX Name:
  DISPLAY Customer.
END.

FOR EACH Customer NO-LOCK WHERE Customer.Balance < 10000
  AND Customer.Balance > 5000:
  DISPLAY Customer.
END.
```

  On the other hand, the following queries do not result in WHOLE-INDEX searches because the selection criteria directly limit the range of `Name` and `CustNum` index keys (respectively) to be searched:

```
FOR EACH Customer NO-LOCK
  WHERE Customer.Name < "Penan Sporttiklubi"
    AND Customer.Name > "Chip's Poker":
  DISPLAY Customer.
END.

FOR EACH Customer NO-LOCK WHERE Customer.CustNum < 40:
  DISPLAY Customer.
END.
```

- For SpeedScript, the following options are invalid: V6FRAME, USE-REVVIDEO, and USE-UNDERLINE.
- When ABL compiles a procedure or class definition file it generates a unique signature value based on the code content, and stores it in the r-code file. You can read the SIGNATURE-VALUE attribute on the RCODE-INFO handle to determine the signature value for a procedure or class. This signature value is used by WebClient to determine if an r-code file has changed since the previous version of the application.
- The COMPILE statement is not synchronized with other ABL sessions. As such, compiling individual files in a common code base when applications are using that same code base for either compilation or execution, can cause unpredictable behavior and is not advised. The best practice is to holistically compile a project

in one ABL session prior to updating a running application. The Progress Application Server (PAS) for OpenEdge supports the ability to easily update an application in production.

For more information, see "Update PROPATH in a production instance with zero downtime" in *Manage Progress Application Server (PAS) for OpenEdge*.

See also

COMPILER system handle, NEW function (classes), NO-ERROR option, RUN statement, Compile Warning List (`-cwl`), Keyword Forget List (`-k`), and No Lock (`-NL`) startup parameters (in *Startup Command and Parameter Reference*)

CONNECT statement

Establishes a connection to one or more databases from within an ABL procedure or class.

Note: OpenEdge identifies all connected databases for access within an external procedure or class at the start of execution for each compilation unit. Therefore, you cannot **directly** connect a database using this statement and **directly** access tables in the database from within the same external procedure or class. Instead, you can directly connect the database in one procedure or class, and from this one call another external subprocedure, or instantiate another class, that accesses the database tables.

Syntax

```
CONNECT
  {
    { physical-name | VALUE ( expression ) } [ options ] | options
  }
  [ NO-ERROR ]
```

physical-name

The actual name of the database on disk. It can be a simple filename, relative pathname, or a fully qualified pathname, represented as an unquoted string, or a quoted string. If you do not give a fully qualified pathname, the AVM searches for the database relative to your current working directory. The database name is restricted to alphanumeric characters. Diacritical marks and the symbols \ " ' * ; | ? [ ] ( ) ! { } < > @ + = : ~ are not permitted.

VALUE (*expression*)

A character expression (a quoted string, field name, variable name, or similar expression) whose value starts with the Physical Database Name (`-db`) connection parameter followed by zero or more of the same client connection parameters that you can specify in *options*.

options

One or more client connection parameters (unquoted), similar to those used to start OpenEdge. Valid options are a subset of OpenEdge startup parameters that include all client database connection parameters. If you specify *options* without *physical-name* or VALUE (*expression*), the first database connection parameter must be the Physical Database Name (`-db`) parameter. The

specification of the User ID (-U) parameter (and Password (-P) parameter, if required), determines the user identity for the connection, and its tenancy (if the database is multi-tenant). Note that these (and all connection) parameters are case sensitive.

If you need to use an argument with the special characters (diacritical marks and the symbols \ " ' * ; | ? [ ] ( ) ! { } < > @ + = : ~), the argument must be quoted for the AVM to handle it properly. The following code shows one way to apply this for a password, vpasswd, that may contain special characters:

```
connString = "-db test -U " + vUser + " -P " + QUOTER(vpasswd).
CONNECT VALUE(connString)
```

Caution: If you do not specify -U and -P, for backward compatibility, OpenEdge attempts to connect the database with a default connection identity. This default connection identity can be set using either the blank ("") user ID or the user ID of the operating system process in which the AVM is running. The user ID set for the default connection identity depends on the domain configuration in the database. For more information on connecting with a default identity, see the User ID (-U) parameter description in *Startup Command and Parameter Reference*. For more information on the effects of connecting a database with a default identity, see the Notes of this statement entry.

For more information on all database client connection parameters, see *Startup Command and Parameter Reference*.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

For the CONNECT statement with NO-ERROR, the option does not suppress all errors produced by the server; only errors caused by the CONNECT statement itself. For example, if the server to which you are connecting runs out of resources, its error message will not be suppressed. If a CONNECT error occurs (for example, the database does not exist or is in use in single-user mode), error information is written to the ERROR-STATUS system handle.

Examples

This procedure attempts to connect to databases mydb1 and mydb2 in single-user mode, with error suppression. You must connect to a database before you run a procedure that references it.

r-connct.p

```
CONNECT mydb1 -1 -db mydb2 -1 NO-ERROR.
```

The following four code fragments attempt exactly the same database connection to the Sports2020 database:

```
CONNECT C:\OpenEdge\WRK\db\Sports2020 -H dbserver -S 1900 NO-ERROR.
```

```
CONNECT -db C:\OpenEdge\WRK\db\Sports2020 -H dbserver -S 1900
 NO-ERROR.
```

```
CONNECT VALUE("-db C:\OpenEdge\WRK\db\Sports2020 -H dbserver
   -S 1900")
 NO-ERROR.
```

```
CONNECT VALUE("-db C:\OpenEdge\WRK\db\Sports2020 -H dbserver")
   -S 1900
  NO-ERROR.
```

The following procedure fragment shows how you can use the VALUE option to specify a user ID (cUserID) and password (cPasswd) that a user might enter in response to a prompt to authenticate the same database connection:

```
DEFINE INPUT PARAMETER cUserID AS CHARACTER NO-UNDO.
DEFINE INPUT PARAMETER cPasswd AS CHARACTER NO-UNDO.

CONNECT C:\OpenEdge\WRK\db\Sports2020
  VALUE( "-U " + cUserID +
        " -P " + "oech1::" + AUDIT-POLICY:ENCRYPT-AUDIT-MAC-KEY(cPasswd))
  -H dbserver -S 1900 NO-ERROR.
```

Note also that this fragment encrypts the password value (cPasswd) and concatenates it with a prefix in a form that OpenEdge expects for encrypted passwords. For more information, see the ENCRYPT-AUDIT-MAC-KEY( ) method reference entry.

In the next example, assume database Sports2020 has not been previously connected, so the following r-cnct1.p procedure fails. At the start of execution, r-cnct1.p checks whether Sports2020 is connected. If Sports2020 is not connected, a run-time error occurs. As shown in the example, attempting to connect to Sports2020 within the procedure does not solve the problem:

r-cnct1.p

```
/* NOTE: this code does NOT work */

CONNECT Sports2020 -1.
FOR EACH Sports2020.Customer NO-LOCK:
  DISPLAY Customer.
END.
```

Instead, split r-cnct1.p into two procedures, as shown in r-dispcu.p and r-cnct2.p:

r-dispcu.p

```
FOR EACH Sports2020.Customer NO-LOCK:
  DISPLAY Customer.
END.
```

r-cnct2.p

```
CONNECT Sports2020 -1.
RUN r-dispcu.p.
```

This time, database `Sports2020` is connected before `r-dispcu.p` is invoked, so `r-dispcu.p` runs successfully.

The next example shows how to use the alternate database connection set parameters (`-dbalt1`, `-dbalt2`, `-retryConnect`, and `-retryConnectPause`) to specify alternate databases to try connect to, in the event that the connection to the primary database fails. Notice how outer double quotes are used to enclose the entire string and inner single quotes are used around the `-dbalt1` and `-dbalt2` arguments. You may also reverse them and use single quotes for the outer connection string and double quotes for the inner parameter values.

```
connString = "-db myDb -H PrimarySrvr -S 6999 -ct 0
  -dbalt1 'myDbBackup -H AltSrvr1 -S 7999'
  -dbalt2 'myDbBackup -H AltSrvr2 -S 7999'
  -retryConnect 10 -retryConnectPause 4".
CONNECT VALUE(connString).
```

Notes

- The user identity set for a database connection determines:
  - If and how the user has permission to access tables and fields in the connected database
  - The tenant organization through which the user access a multi-tenant database
  - The audit identity used to record audit policy events during the connection process
- OpenEdge authenticates any user identity that you specify for the `CONNECT` statement using the local database domain registry, even if the database option is set to use the application (session) domain registry.
- To authenticate the user identity specified for a database connection:
  - The user's security domain must be defined in the OpenEdge database.
  - The domain must be authentication-enabled:
    1. The user's domain must be enabled in the database.
    2. The user's domain must be configured with an authentication system that supports OpenEdge-performed user authentication. For the `CONNECT` statement (but **not** the startup command line), this can include a domain configured with a user-defined authentication system that has an ABL authentication plugin enabled.
    3. The configured authentication system must have access to a source of valid user accounts.
  - The user credentials specified by the `-U` and `-P` parameters must match a user account accessible through the authentication system configured for the user's domain.

For information on OpenEdge support for domains and domain configuration, see *Introduction to Identity Management*.

- The user authentication operation of the CONNECT statement can fail for any one of the following reasons, among others:
  - The User ID (-U) connection parameter includes an invalid format.
  - The user ID's account is not found in the domain.
  - The domain is not defined.
  - The domain is not enabled.
  - The Password (-P) value specified for the user account is invalid.
  - The domain is configured for single sign-on (SSO) operations only.
  - The authentication system returns an error for any other reason.
- For the connection identity set with the CONNECT statement, OpenEdge creates a sealed security token containing the user credentials for the database connection, which you can return as a client-principal object using the GET-DB-CLIENT function. This client-principal is created even if OpenEdge connects the database with a default connection identity (that is, you do not specify -U and -P). However, for a default connection identity, OpenEdge does not seal the client-principal using the access code configured for a registered domain. Instead, OpenEdge creates a unique internal access code to seal the object. As a result, you cannot use the sealed client-principal object to assign the default user identity it represents to any OpenEdge database connection or ABL session.
- Each connected database is assigned a logical name for the current session, and is referred to by this logical name during the session. Use the Logical Database Name (-ld) parameter to specify a logical name. If the logical name is not specified using the -ld parameter, then the physical database filename, without the .db suffix, is the default logical name. For example, if the physical name is /users/eastcoast/proapp/mydb.db, then the default logical name is mydb. Logical names are **not** case sensitive.
- Databases can have aliases (see also ALIAS function). A database can have more than one alias, but each alias refers to only one database. The first database connected during a given session automatically receives the alias DICTDB. The first database connected that has a _menu file automatically receives the alias FTDB. You can reassign the FTDB alias to any other FAST TRACK database.
- When you try to connect the same database twice using the same logical name, the AVM returns a warning, which you can suppress with NO-ERROR.
- When you try to connect different databases using the same logical name, the AVM returns an error message and an error condition. You can suppress the error condition with NO-ERROR, and test with the CONNECTED function.
- When you try to connect to multiple databases and a connection fails, a run-time error occurs. The successfully connected databases remain connected and program execution continues. Use the CONNECTED function to find out which databases are successfully connected.
- If you run a procedure that requires a database and that database is not connected, the AVM searches for the database in the auto-connect lists in all connected databases. If the AVM finds the required database there, it automatically attempts to connect to the database with the parameters set for it in the auto-connect list. You can edit the auto-connect list using the database utilities in the OpenEdge Data Dictionary. If the AVM does not find it, the connection attempt fails.
- Connection information found in an OpenEdge auto-connect list is merged with connection information in a CONNECT statement that connects the database. So, if you connect a database with a CONNECT statement, and that database already has an entry in the OpenEdge auto-connect list of a connected database, the

connection information in the auto-connect list and the CONNECT statement is merged. However, the connection information in the CONNECT statement takes precedence.

- Permission issues limit the use of the CONNECT statement for raw I/O connections to databases in single-user and multi-user direct-access mode on UNIX machines that do not support O_SYNC and SWRITE.

  The ABL client executable might require use of a privileged account that allows it to open raw disk devices or large databases. Thus, you can open any databases specified on the startup command line with raw I/O. Note that after startup on Unix, the client executable relinquishes the privileges that allow it to open raw disk devices. As a result, you cannot use the CONNECT statement to establish a raw I/O connection to a database in single-user or multi-user direct-access mode.

  When you try to use a CONNECT statement to open a raw I/O connection to a database in single-user mode, the AVM establishes a buffered (non-raw) I/O connection to the database and displays a non-raw warning message.

- When you try to use a CONNECT statement to open a raw I/O connection to a database in multi-user direct-access mode, one of the following events occur:

  - If you started a server (PROSERVE) for the database with the Buffered I/O (-r) parameter, the AVM establishes a non-raw I/O connection to the database.
  - If you started a server (PROSERVE) for the database with the Raw I/O (-R) parameter, the CONNECT statement fails.

  There are several ways to avoid these problems:

  - Establish raw I/O database connections in the single-user and multi-user direct-access modes at ABL startup.
  - If you must use the CONNECT statement to establish a raw I/O database connection, establish the connection with the Client Multi-user (-cl) parameter. Be sure to start the database server (PROSERVE) with the Raw I/O (-R) parameter before you do this.
  - If you must use the CONNECT statement to establish a raw I/O database connection in single-user or multi-user direct access mode on UNIX, follow these steps **carefully**:

    1. Change the permissions of the ABL client executable to rwsrwsr-x by typing **chmod 6775 _progres**.
    2. Change the group of the client executable to match the group of the raw device (for example, /dev/rsd0d) and block special device (for example, /dev/sd0d).
    3. Change the permissions of the raw and block special devices to "rw-rw----".

    The disadvantage of this procedure is that all files produced within OpenEdge have the same group as the disk device. Consider the following:

  - If you want to run a multi-user direct-access session in non-raw mode, you must start the database server with the Buffered I/O (-r) parameter.
  - If a database and accompanying before-image file have read-only permissions (r--r--r--) and you try to connect to that database in single-user or multi-user mode using the CONNECT statement, the connection will fail with the following error:

    ```
    errno=13
    ```

    This connection failure results because the _progres module relinquishes superuser privileges after startup and no longer possesses the privileges required to connect to the database using the CONNECT statement.

- This statement does not attempt set the connection identity for the foreign data source of a DataServer connection. However, it does attempt to set the connection identity for the OpenEdge schema holder database.
- For more information on connecting to databases from ABL, see *OpenEdge Programming Interfaces*.

See also

ALIAS function, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NO-ERROR option, NUM-DBS function, PDBNAME function, SDBNAME function, SET-DB-CLIENT function, SETUSERID function

CONNECTED function

Tells whether a database is connected. If *logical name* is the logical name or *alias* is the alias of a connected database, the CONNECTED function returns TRUE; otherwise, it returns FALSE.

Syntax

```
CONNECTED ( logical-name | alias )
```

logical-name

Refers to a logical name. It can be a quoted string or a character expression. An unquoted character string is not allowed.

alias

Refers to an alias. It can be a quoted string or a character expression. An unquoted character string is not allowed.

Example

This procedure runs r-dispcu.p if a database with the logical name Sports2020 is connected:

r-cnctd.p

```
IF CONNECTED("Sports2020") THEN RUN r-dispcu.p.
```

Notes

- By default the AVM disconnects all databases at the end of a session. The DISCONNECT statement, which explicitly disconnects a database, does not execute until all active procedures that reference the database end or stop. Therefore the disconnection may happen later than the end of a transaction.

See also

ALIAS function, CONNECT statement, CREATE ALIAS statement,CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

CONSTRUCTOR statement

Defines a constructor for a class. A constructor is a special type of method that ABL invokes to initialize data for a new object of a class that is instantiated using the NEW function (classes), NEW statement, or DYNAMIC-NEW statement, or to initialize static members of a class.

Note: This statement is applicable only when used in a class definition (`.cls`) file.

Syntax

```
CONSTRUCTOR [PRIVATE | PACKAGE-PRIVATE | PROTECTED | PACKAGE-PROTECTED | PUBLIC] [STATIC]
  class-name ( [ parameter [ , parameter ] ... ] ) :
  constructor-body
```

`[PRIVATE | PACKAGE-PRIVATE | PROTECTED | PACKAGE-PROTECTED | PUBLIC]`

Specifies the access mode for the constructor.

A `PRIVATE` constructor can be called by the defining class or another instance of the same class. This is done via the THIS-OBJECT statement within another constructor of the class or using the `NEW` statement or function anywhere within the class. To create the first instance of a class with only a private constructor, you can call a static method in the class which will use `NEW` to create an instance.

A `PACKAGE-PRIVATE` constructor can be called from the defining class or any class within its package. This is done via the SUPER statement within another constructor or using the `NEW` statement or function elsewhere.

A `PROTECTED` constructor can be called from the defining class or a subclass. This is done via the SUPER statement within another constructor or using the NEW statement or NEW function (classes) elsewhere.

A `PACKAGE-PROTECTED` constructor can be called from the defining class, from a subclass, or from any class within its package. This is done via the SUPER statement within another constructor or using the `NEW` statement or function elsewhere.

A `PUBLIC` constructor can be called from any code (in a class or procedure) that has access to the class instance. This is done via the SUPER statement within another constructor or using the `NEW` statement or function elsewhere.

The default access mode is `PUBLIC`.

[STATIC]

STATIC specifies a static constructor that executes exactly once in an ABL session the first time you reference a class type that defines this constructor in its class hierarchy. You cannot invoke a static constructor in any other way or at any other time. You can define only one static constructor for a given class. If you do not define a static constructor, ABL defines a default static constructor to initialize the static members of a class.

class-name

The name of the class this method constructs. This name must match the class name portion of the type name for the class (that is, the name of the class definition file excluding the .cls extension and any package path information).

([*parameter* **[** *, parameter* **]...]**)

Optionally specifies one or more parameters of the constructor. Any instance constructor defined without a parameter list is the default instance constructor for the defining class. The parameter list for a defined static constructor must be empty.

If this instance constructor is one of several overloaded constructors defined for a class, the parameter list must be unique among all the other constructors. This uniqueness can be established using a different combination of number, data types, or modes for the parameters. For information on the parameter definition syntax and establishing uniqueness for overloaded constructors, see the Parameter definition syntax reference entry. Note that any defined static constructor does not participate in constructor overloading with instance constructors.

constructor-body

The body of the constructor definition. Define the constructor body using the following syntax:

```
constructor-logic
. . .
END [ CONSTRUCTOR ].
```

constructor-logic

The logic of the constructor, which can contain any ABL statements currently allowed within a PROCEDURE block including class-related statements. These statements typically contain logic to initialize the data members and properties in the class.

Each logic statement must end with a period.

If you are defining an instance constructor, regardless of other statements in the constructor, the first action of the constructor must be a call to another instance constructor in the defining class or in the immediate super class. ABL can call a default super class constructor implicitly, or you can call a super class constructor or another overloaded constructor in the defining class explicitly as the first statement of a constructor. You cannot explicitly call another constructor in any other statement of a constructor, and you cannot call any defined static constructor.

If there is no constructor instance defined in the immediate super class and you do not explicitly invoke a constructor, ABL always implicitly invokes the built-in default super class constructor (without parameters) as the first action. If there is an instance constructor defined in the super class that does not take parameters, you also do not need to explicitly invoke an instance constructor. ABL implicitly invokes this user-defined default super class constructor. You only need to explicitly invoke another instance constructor when the super class has constructors defined for it and all of these constructors take parameters.

When you invoke an instance constructor that takes parameters, again, you must invoke that constructor as the first executable statement in the invoking constructor. If you want to invoke a super class instance constructor, you must invoke the SUPER statement with parameters that match the parameters of the super class constructor with respect to number, data type, and mode.

If you want to invoke an overloaded instance constructor of the defining class, you must invoke the THIS-OBJECT statement as the first statement, with parameters that match the parameters of the overloaded constructor with respect to number, data type, and mode. If you invoke an overloaded constructor, and that overloaded constructor does not invoke another overloaded constructor, it must invoke a super class constructor, either implicitly (the default) or explicitly as its first statement. So, in any chain of explicit calls from one overloaded constructor to another, the last overloaded constructor in the chain must invoke a super class constructor.

If you are defining a static constructor, you cannot access any instance members of a class (including the defining class), nor can you use the SUPER and THIS-OBJECT statements. From a static constructor, you can access only other static members of a class and the local variables or other local data elements of the constructor.

```
END [ CONSTRUCTOR ]
```

Specifies the end of the constructor body definition. You must end the constructor body definition with the END statement.

Examples

The following example shows the definition of an instance constructor:

```
CONSTRUCTOR PUBLIC CustObj( ):

  m_NumCusts = 0.

  /* Fill a temp table and get the row count */
  FOR EACH Customer NO-LOCK:
    CREATE ttCust.
    ASSIGN
      ttCust.CustNum = Customer.CustNum
      ttCust.Name    = Customer.Name
      m_NumCusts     = m_NumCusts + 1.
  END.

END CONSTRUCTOR.
```

For more examples of constructor definitions, including a static constructor and a constructor for an abstract class, see the descriptions of r-CustObj.cls, r-CustObjStatic.cls, and r-CustObjAbstract.cls in the CLASS statement reference entry.

Notes

- You can terminate a CONSTRUCTOR statement with either a period (.) or a colon (:), but typically use a colon (:).
- A constructor definition must begin with the CONSTRUCTOR statement and end with the END statement.
- A constructor has no return type.
- You never explicitly invoke an instance constructor to create a class instance. The constructor is implicitly invoked when you instantiate the defining class using the NEW function (classes), NEW statement, or DYNAMIC-NEW statement, passing any parameters required to identify the instantiating constructor. The instantiating constructor then directly or indirectly calls an instance constructor in its immediate super class, which similarly calls an instance constructor in its immediate super class, and so on for all classes in the class hierarchy, until the default instance constructor of the root (Progress.Lang.Object class) is called. From this point, the root constructor and all previously called instance constructors complete execution in reverse order of invocation, terminating class instantiation with completion of the initial instantiating constructor.
- Within a class hierarchy, you can explicitly invoke an instance constructor from an instance constructor of an immediate subclass using the SUPER statement or from an overloaded constructor in the same defining class using the THIS-OBJECT statement. The invoking constructor must specify any parameters required to identify the called constructor. These parameters must match the constructor parameters with respect to the number, data type, and mode.
- You never explicitly invoke a static constructor. All the static constructors of a class hierarchy with static members execute on first reference to a given class within an ABL session. This first reference causes the static constructors of all super classes in the referenced class hierarchy that have static members to execute from top to bottom, terminating with the execution of the static constructor for the most derived class with static members. For any single class with static members, its static constructor runs only once per ABL session or until after the given class is compiled again. If the first reference to a class occurs during class instantiation, this sequence of static constructor execution occurs prior to execution of the instantiating constructor. Thus, all static constructors in a class hierarchy are guaranteed to execute before a class instance is referenced.
- Prior to execution of a given static constructor, all of the static data members and properties of the defining class are set to their initial values. The constructor then accesses these initial values of static members, as appropriate. This initialization of static members occurs only once per session or until the class is recompiled.
- You can handle application errors in an instance constructor as in any ABL block. However, by executing a RETURN ERROR action at the block level or a THROW action at the block level with the presence of a BLOCK-LEVEL ON ERROR UNDO, THROW statement, or a ROUTINE-LEVEL ON ERROR UNDO, THROW statement, the AVM returns the ERROR condition from the constructor block. With this returned ERROR condition, ABL terminates creation of the object. If any part of the class hierarchy has been created (constructors executed), ABL executes the corresponding destructors for those classes automatically and raises ERROR on the statement that instantiated the class. If this constructor is invoked as part of a NEW statement or DYNAMIC-NEW statement, the data element set to receive an object reference to the failed class instantiation remains unchanged. If a RETURN ERROR also includes the option to return a character string value, or you set the ReturnValue property of a Progess.Lang.AppError object that you THROW, you can get this value using the RETURN-VALUE function following the statement that attempted to instantiate the class or in a CATCH block that catches the Progress.Lang.AppError object. For more information, see *Develop Object-oriented ABL Applications*.
- You can handle application errors in a static constructor similar to an instance constructor. However, an ERROR condition returned from the constructor block is raised on the statement whose class reference caused the static constructor to be invoked (whether or not the statement instantiates the class). With this ERROR condition, ABL fails to loaded the specified class and its entire class hierarchy. In addition, ABL does not load the specified class, or the other classes of its class hierarchy, for the remainder of the ABL session, or until the specified class is recompiled.

See also

Assignment (=) statement, CLASS statement, DESTRUCTOR statement, DYNAMIC-NEW statement, FUNCTION statement, NEW function (classes), NEW statement, Parameter definition syntax, SUPER statement, THIS-OBJECT statement, VAR statement

COPY-LOB statement

Copies large object data between BLOBs, CLOBs, MEMPTRs, and LONGCHARs. It also copies large object data to and from the file system, and converts large object data to or from a specified code page.

Note: You cannot copy large object data between BLOBs and CLOBs directly. However, you can copy a BLOB or CLOB to a MEMPTR or LONGCHAR (which converts the data) and then copy the MEMPTR or LONGCHAR to the CLOB or BLOB, respectively.

Syntax

```
COPY-LOB
  [ FROM ] { [ OBJECT ] source-lob | FILE source-filename }
     [ STARTING AT n ] [ FOR length ]
  TO { [ OBJECT ] target-lob [OVERLAY AT n [TRIM ] ] |
     FILE target-filename [ APPEND ] }
  [ NO-CONVERT | CONVERT convert-phrase ]
  [ NO-ERROR ].
```

[OBJECT]*source-lob*

The source object to be copied, which can be a MEMPTR or LONGCHAR variable, a BLOB or CLOB database or temp-table field, or a dynamic expression that resolves to a BLOB or CLOB database or temp-table field. The source object data at this location is copied to the specified target object or file.

FILE *source-filename*

A character expression that specifies the name of a file containing the source object data to be copied. The object data in this source file is copied to the specified target object or file. You can specify an absolute or relative pathname. Any relative pathname is relative to the current working directory.

The AVM raises the ERROR condition if *source-filename* resolves to the Unknown value (?) or the source file cannot be read. The filename can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

The AVM raises an error if both *source-filename* and *target-filename* are specified and they resolve to the same file.

STARTING AT *n*

An integer expression indicating a one-based offset position, in the source object or file, from which to start copying. The copy begins at offset 1, by default. The AVM raises the ERROR condition if the

specified offset position is less than 1, greater than the size of the object or file, or the Unknown value (?).

Note: Offsets are measured in bytes for binary data (BLOB or MEMPTR), and characters for character data (CLOB or LONGCHAR).

FOR *length*

An integer expression indicating the number of bytes or characters to copy from the source object or file starting at the specified offset position. The AVM copies from the specified offset position to the end of the object or file, by default. The AVM raises the ERROR condition if the specified length is less than 0, greater than the size of the object or file, or the Unknown value (?).

Note: Offsets are measured in bytes for binary data (BLOB or MEMPTR), and characters for character data (CLOB or LONGCHAR).

[OBJECT] *target-lob*

The target object to receive the copy, which can be a MEMPTR or LONGCHAR variable, a BLOB or CLOB database or temp-table field, or a dynamic expression that resolves to a BLOB or CLOB database or temp-table field. The object data in the specified source object or file is copied to the target object.

If the specified target object does not yet exist, the AVM either creates a BLOB or a CLOB, or allocates memory for a MEMPTR or a LONGCHAR. If the specified target object already exists, the AVM deletes the object before the copy operation begins, by default. You can specify the OVERLAY AT *n* option to overlay some portion of an existing target object.

Note: Although the AVM allocates memory for a target MEMPTR, you are responsible for freeing that memory.

OVERLAY AT *n* [TRIM]

An overlay position in the target object. The AVM copies the source object or file to an existing BLOB, CLOB, MEMPTR, or LONGCHAR target starting at the given position. If the operation results in writing past the end of a target BLOB, CLOB, or LONGCHAR, the AVM extends the target object as necessary. If the operation results in writing past the end of a target MEMPTR, the AVM raises the ERROR condition.

If the target object does not yet exist, the AVM raises the ERROR condition. If the specified overlay position is less than 1, greater than the size of the object, or the Unknown value (?), the AVM raises the ERROR condition.

You can specify the TRIM option only if the target object is a BLOB or CLOB. In this case, the AVM copies the source object or file to the existing target object and truncates any data remaining in the target object. If the target object is a MEMPTR, the AVM ignores this option.

FILE *target-filename* [APPEND]

A character expression that specifies the name of the target file to which the object data in the specified source object or file is copied. You can specify an absolute or relative pathname. Any relative pathname is relative to the current working directory.

If the target file does not exist, the AVM creates the file. If the target file exists, and you specify the APPEND option, the AVM opens the file and appends the object data to the end of a file. If the target file exists, but you do not specify the APPEND option, the AVM creates the target file anew (which overwrites the original file).

If *target-filename* resolves to the Unknown value (?), or the target file cannot be created or written, the AVM raises the ERROR condition. The filename can contain Unicode characters.

The AVM raises an error if both *source-filename* and *target-filename* are specified and they resolve to the same file.

NO-CONVERT | CONVERT *convert-phrase*

Lets you specify the character conversion behavior between the source and target objects.

The NO-CONVERT option specifies that no conversions occur. However, if the target is a LONGCHAR or a CLOB, the AVM validates the character data based on the target object's code page. For a CLOB, this is the code page of the CLOB. For a LONGCHAR, this is -cpinternal unless the LONGCHAR's code page was set using the FIX-CODEPAGE statement. If the validation fails, the AVM raises the ERROR condition.

The CONVERT option lets you specify how the AVM converts object data. Following is the syntax for *convert-phrase:*

```
{
  [ SOURCE CODEPAGE codepage ]
  [ TARGET CODEPAGE codepage ]
}
```

Specify SOURCE CODEPAGE to indicate that a source object is in the specified code page. If you specify TARGET CODEPAGE, the AVM converts the target object to the specified code page.

The following table lists the default character conversions the AVM performs when copying data between the source and target objects. References to CLOBCP and CLOBDB represent CLOB data in either the CLOB's defined code page or the database's defined code page, respectively. References to the "fixed code page" represent the code page of a target LONGCHAR variable set using the FIX-CODEPAGE statement.

Table 19: Default COPY-LOB statement character conversions

When the source object is a . . .	And the target object is a . . .	The AVM converts the source object . . .
MEMPTR	LONGCHAR	From -cpinternal to -cpinternal or the fixed code page
MEMPTR	CLOBDB	From -cpinternal to the database's defined code page
MEMPTR	CLOBCP	From -cpinternal to the CLOB's defined code page
BLOB	LONGCHAR	No conversion, the LONGCHAR is in -cpinternal or the fixed code page

When the source object is a . . .	And the target object is a . . .	The AVM converts the source object . . .
LONGCHAR	MEMPTR	From the LONGCHAR's code page to `-cpinternal`
LONGCHAR	BLOB	No conversion, the BLOB's code page is unknown
LONGCHAR	CLOBDB	From the LONGCHAR's code page to the database's defined code page
LONGCHAR	CLOBCP	From the LONGCHAR's code page to the CLOB's defined code page
CLOBDB	MEMPTR	From the database's defined code page to `-cpinternal`
CLOBDB	LONGCHAR	From the database's defined code page to `-cpinternal` or the fixed code page
CLOBCP	MEMPTR	From the CLOB's defined code page to `-cpinternal`
CLOBCP	LONGCHAR	No conversion, or conversion to the fixed code page

Note:

If either the source or the target object is a file, the target's code page defaults to `-cpstream`.

The Byte Order Mark (BOM) or the encoding declaration is processed when a `COPY-LOB` operation reads from or writes to a FILE. For example, when a `COPY-LOB` statement copies data from a FILE (with a BOM) to a CLOB, the statement identifies and translates file content based on the BOM character but ignores the BOM character as part of the copy operation. And, when a `COPY-LOB` operation copies data from a CLOB to the beginning of a FILE, the statement prepends the BOM character, except when the BOM can be assumed; that is, the code page of the FILE target is UTF-8.

Also, if the `COPY-LOB` operation is written to a `STARTING-AT` location embedded within the FILE, the BOM character is not prepended to the written content. For more information on encoding declarations, see the *Use XML with ABL Applications* guide.

`NO-ERROR`

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Notes

- If a source or target object is stored in a database, its record must be available to copy. The lock mode of the record containing the target object must be EXCLUSIVE-LOCK or SHARE-LOCK and upgradeable; otherwise, the `COPY-LOB` statement raises the ERROR condition.
- You can also assign large object data from one BLOB or MEMPTR to another, and one CLOB or LONGCHAR to another, using the Assignment (=) statement or ASSIGN statement. You cannot use the Assignment (=) statement or `ASSIGN` statement to assign large object data between BLOBs or MEMPTRs and CLOBs or LONGCHARs.
- COPY-LOB file operations involving LONGCHAR variables are limited to 1GB - 1 byte.

See also

NO-ERROR option

COUNT-OF function

Returns an INTEGER value that is the total number of selected records in the file or files you are using across break groups.

Syntax

```
COUNT-OF ( break-group )
```

break-group

The name of a field or expression you named in the block header with the BREAK BY option.

Example

This procedure sorts all customers by state and then calculates the percentage of the total number of customers that are in each state. The COUNT-OF function provides the calculation with the number of customer records in the database.

r-cntof.p

```
FOR EACH Customer NO-LOCK BREAK BY Customer.State:
  DISPLAY Customer.CustNum Customer.Name Customer.SalesRep Customer.State.
  ACCUMULATE Customer.State (SUB-COUNT BY Customer.State).
  IF LAST-OF( Customer.State) THEN
    DISPLAY 100 * (ACCUM SUB-COUNT BY Customer.State Customer.State) /
      COUNT-OF(Customer.State) FORMAT "99.9999%"
      COLUMN-LABEL "% of Total!Customers".
END.
```

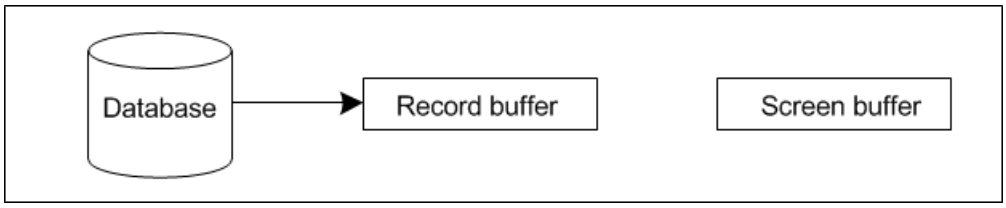
See also

Aggregate phrase

CREATE statement

Creates a record in a table, sets all the fields in the record to their default initial values, and moves a copy of the record to the record buffer.

Data movement



Syntax

```
CREATE record [ FOR TENANT tenant-expression ]
[ USING { ROWID ( nrow ) | RECID ( nrec ) } ] [ NO-ERROR ]
```

record

The name of the record or record buffer you are creating.

To create a record in a table defined for multiple databases, you might have to qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

FOR TENANT *tenant-expression*

This option is useful only for a multi-tenant database, and primarily one with a connection identity that has super tenant access. If the user has a super-tenant connection identity and you **do not** specify this option, the record you create is owned by the effective tenant. If you **do** specify this option, you create a record owned by the regular tenant identified by *tenant-expression*.

If the user has a regular-tenant connection identity, and you specify this option, *tenant-expression* must match the tenancy of the connection identity. Otherwise, the statement raises ERROR.

If *tenant-expression* evaluates to an integer, the value must be a valid tenant ID for a regular tenant or zero (0) for the default tenant. If *tenant-expression* evaluates to a character string, the value must be a valid tenant name for a regular or "Default" for the default tenant. Otherwise, the statement raises ERROR.

If *record* belongs to a table that is not multi-tenant enabled, ABL raises a compiler error.

USING { ROWID (*nrow*) | RECID (*nrec*) }

Supported only for backward compatibility.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

The following example creates a record in the order file for each pass through the loop and then updates the record. It also creates an order-line record.

r-create.p

```
REPEAT:
  CREATE Order.
  UPDATE Order.OrderNum Order.CustNum
    VALIDATE(CAN-FIND(Customer OF Order), "Customer does not exist")
    Order.CustNum Order.OrderDate.
  REPEAT:
    CREATE OrderLine.
    OrderLine.OrderNum = Order.OrderNnum.
    UPDATE OrderLine.LineNum OrderLine.ItemNum
      VALIDATE(CAN-FIND(Item OF OrderLine), "Item does not exist")
      OrderLine.Qty OrderLine.Price.
  END.
END.
```

This procedure adds `Orders` and `OrderLines` to the database. Because the user supplies an order number when updating the `order` record, that order number is assigned (=) to the `OrderNum` field of the `OrderLine` record when the `OrderLine` record is created.

Notes

- When you run procedures that create large numbers of records (for example, during initial data loading), the process runs much faster if you use the No Crash Protection (-i) parameter. See *Startup Command and Parameter Reference* for more information on startup parameters. Back up your database before you use this parameter.
- After you create a new record with `CREATE`, the AVM waits to write the record to the database. The creation often happens after one of the following:
  - The AVM is about to release the buffer, which can happen as a result of an explicit `RELEASE` or `VALIDATE` statement, the reading of a new record into the buffer, the record going out of scope, or the end of the transaction.
  - You set a LOB field.
  - You set a field that an index is based on.
- All partition fields must be set by the time a record is created in a partitioned database.

  Suppose you have an `Order` table partitioned on `OrderDate`, but with a global index on `OrderNum`. If `OrderDate` has an initial value of the Unknown value (?), the following code will produce an error:

  ```
  CREATE Order.
  OrderNum = NEXT-VALUE(Next-Ord-Num).
  OrderDate = TODAY.
  ```

  After the assignment of the `OrderNum` field, the AVM will try to index the new `OrderNum` in the `OrderNum` global index. To do that, it needs the partition of the new `Order` record. To get the partition, the AVM needs to use the partition column `OrderDate`, which is still the Unknown value (?). This will produce an error message stating that `OrderDate` is not in any defined partition. You can fix this by reversing the order of the assignments, grouping the statements into one `ASSIGN` statement, or setting an initial value for the `OrderDate` column. If you use one `ASSIGN` statement, the actual create of the record won't occur until after all the specified field assignments are gathered. Therefore the `OrderDate` column will already have the correct value by the time the partition for the new record needs to be determined.

Note: Even if all of the partition fields have initial values, an application might be inefficient if the underlying record creation happens before the partition fields are set to something other than the initial values. In this case, the AVM may have to move the record to a new partition once the initial value of the partition field is replaced.

- The CREATE statement causes any related database CREATE triggers to execute. All CREATE triggers execute after the record is actually created. If a CREATE trigger fails (or executes a RETURN statement with the ERROR option), the record creation is undone.
- When specifying the FOR TENANT option, the AVM looks up *tenant-expression* in the database with a share lock. The AVM waits 60 seconds to get the share lock and raises ERROR if it fails to obtain the share lock in that amount of time. The AVM releases the share lock immediately after successfully fetching the row. This share lock is released even if the statement is called while in the scope of a transaction.

See also

BUFFER-CREATE( ) method, INSERT statement, NEW function (record buffers), NO-ERROR option

CREATE ALIAS statement

Creates an alias for a database. Once an alias is created, it can be used in place of the database's logical name.

Note: A database can have more than one alias, but each alias refers to one and only one database.

Syntax

```
CREATE ALIAS alias-string | value ( expression )
  FOR DATABASE logical-name-string | value ( expression )
  [ NO-ERROR ]
```

alias-string | value (*expression*)

An unquoted string, quoted string, or CHARACTER expression that represents an alias for the database.

FOR DATABASE *logical-name-string* | value (*expression*)

An unquoted string, quoted string, or CHARACTER expression that represents the logical name of the database.

Note: The logical name must already be set.

NO-ERROR

Tells the AVM to allow the alias to be created even if the database is not connected.

If you CREATE ALIAS for a database that is not connected and omit NO-ERROR, the AVM reports a run-time error.

Note: The NO-ERROR option of the CREATE ALIAS statement behaves differently from the NO-ERROR option of other ABL elements.

Example

This procedure creates the alias `myalias` for database `mydb`:

r-cralas.p

```
CREATE ALIAS myalias FOR DATABASE mydb NO-ERROR
```

Notes

- The first OpenEdge database connected during a given session receives the DICTDB alias.
- The first database connected that has an `_menu` file automatically receives the alias FTDB. You can reassign the FTDB alias to any other FAST TRACK database.
- If there is already a database connected with logical name equal to *`alias`*, CREATE ALIAS fails.
- If there is an existing alias equal to *`alias`*, the existing alias is replaced by the new alias.
- If you want to use an expression for an alias name or logical name, you must use CREATE ALIAS VALUE (*`expression`*) FOR DATABASE VALUE (*`expression`*).
- When a given database is disconnected, the existing aliases that refer to it are not erased, but remain in the session alias table. Later in the same session, if you connect to a database with the same logical name, the same alias is used again.
- Aliases allow a general purpose application (such as the OpenEdge Data Dictionary) to expect a specific database name. The Dictionary only works on databases with logical name or alias "DICTDB". The end user or the application can use CREATE ALIAS to provide the correct alias, in case it is inconvenient to connect the database using the correct logical name. Also, if there are several connected databases, the application can ask the user which one to select, then set the alias accordingly. The Data Dictionary does this when you choose Select Working Database.
- Suppose you connect to a database with logical name MYNAME and compile a procedure that accesses that database. Normally, the saved r-code file contains references to MYNAME.

  In a later session, when you want to use the precompiled program, you can connect to your database with the same logical name (MYNAME), or you can connect with a different logical name and set up an alias with the statement CREATE ALIAS "MYNAME" FOR DATABASE *`logical-name`*.

- Usually, any alias that exists during the session when you compile a procedure has no effect on the resulting r-code file. When a procedure is compiled, the logical name of the database that is accessed within the procedure is put into the r-code file, not an existing alias. If a procedure accesses more than one database, all of the logical names of accessed databases are placed into the r-code file.

  However, any file reference that is qualified with an alias (as opposed to a logical name) generates a new instance of the file for the compilation. This new instance causes the r-code to have the alias reference and not the logical database name reference. Subsequent unqualified references to that same file within the same block, or nested blocks, will resolve to the new alias instance following the usual rules for qualifying. Unqualified references to different files in the same database do not get the alias name, but get the logical name. Anonymous references to a file, previously referenced using the alias qualifier, in a different, non-nested block get the logical name instead of the alias name.

It is simpler to just connect to a database with the desired logical name, leave all references unqualified, not create an alias, and then compile the application. However, sometimes you cannot precompile. In those cases, if you want to compile a procedure so that only the alias gets into the r-code file, then explicitly qualify all file references using the alias. You might want only the alias to get into the r-code file, so you can compile and distribute procedures that will run against any database whose logical name has been assigned the alias contained in the r-code file.

- Changes made to an alias do not take effect within the current procedure. In the following example, `alias1.p` fails to compile when it reaches the FOR EACH statement, because alias `myalias` has been created during the compilation:

```
/* alias1.p */
/* NOTE: this code does NOT work */
CREATE ALIAS myalias FOR DATABASE Sports2020.

FOR EACH myalias.Customer NO-LOCK:
  DISPLAY Customer.Name.
END.
```

To solve this problem, split `r-alias1.p` into two procedures. For example:

r-dispnm.p

```
FOR EACH myalias.Customer NO-LOCK:
  DISPLAY Customer.Name.
END.
```

r-alias2.p

```
CREATE ALIAS myalias FOR DATABASE Sports2020.
RUN r-dispnm.p.
```

CREATE ALIAS affects only subsequent compilations; currently executing procedures are not affected.

- Be careful when using shared buffers with aliases. If you reference a shared buffer after changing the alias that initially was used in defining it, the AVM returns a run-time error. See the following example procedures for details.

  Once procedure `r-main.p` is run, it calls `r-makebf.p`, which calls `r-disp6.p`. The alias myalias is created in `r-main.p`, with reference to database `Sports2020`. In `r-makebf.p`, the shared buffer mybuf is defined for myalias.customer. Then, in the next line, myalias is changed, so that it now refers to database sports2. When an attempt is made to reference shared buffer mybuf in procedure `r-disp6.p`, a run-time error occurs, with the message: "r-disp6.p Unable to find shared buffer for mybuf."

  r-main.p

```
CREATE ALIAS myalias FOR DATABASE Sports2020.
RUN r-makebf.p.
```

r-makebf.p

```
DEFINE NEW SHARED BUFFER mybuf FOR myalias.Customer.

CREATE ALIAS myalias FOR DATABASE sports2.
RUN r-disp6.p
```

r-disp6.p

```
DEFINE SHARED BUFFER mybuf FOR myalias.Customer.

FOR EACH mybuf NO-LOCK:
  DISPLAY mybuf.
END.
```

- Be careful when accessing a database sequence with an alias that points to a different database than the one used when the alias was defined. If you supply an alias name to the CURRENT-VALUE function or the NEXT-VALUE function, only the database used to define the alias is referenced. In this case, it is preferable to use the DYNAMIC-CURRENT-VALUE function and DYNAMIC-NEXT-VALUE function instead of the CURRENT-VALUE function and NEXT-VALUE function, respectively.

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, DYNAMIC-CURRENT-VALUE function, DYNAMIC-NEXT-VALUE function, ERROR-STATUS system handle, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

---

CREATE automation object statement
(Windows only)

Creates (instantiates) an ActiveX Automation object based on a specified Automation Server connection.

Syntax

```
CREATE expression1COM-hdl-var
[ CONNECT [ TO expression2 ] ]
[ NO-ERROR ]
```

expression1

A character-string expression that evaluates to 1) a unique name of a valid Automation object stored in the system registry or 2) the null string ("").

COM-hdl-var

A COM-HANDLE variable that receives the COM handle to the instantiated Automation object.

[CONNECT **[** TO *expression2* **]** **]**

Specifies the connection option, together with *expression1*. The behavior of each connection option depends on the execution status of the Automation Server. The following table illustrates this behavior.

Table 20: Automation object connection options

Connection option	Server execution status	Connection behavior
1. Option omitted	Running	Creates a new instance of the Automation object identified by *expression1*. Launches a new instance of the Server for top-level Automation objects (like Excel.Application).
	Not running	Launches a new instance of the Server, then creates a new instance of the Automation object identified by *expression1*. Often, both the new Server and the new Automation object instance are invisibly created.
2. CONNECT	Running	Connects to an active (instantiated) Automation object identified by *expression1*. Works for top-level Automation objects only. For example, this works for Excel.Application but fails for Excel.Sheet and Excel.Chart, which are both lower-level Automation objects.
	Not running	Invalid. Always returns an error.
3. CONNECT TO *expression2*	Running	Creates or connects to an Automation object specified by *expression1* that is associated with the file specified by the pathname in *expression2*. If more than one instance of the Server is running, this option randomly selects one (generally, the first one started). If the specified file is already open within the selected Server, this option connects to the Automation object that is instantiated for that file. If the file is not already open in the selected Server, this option opens the file and instantiates the specified Automation object for it. If the specified file is already open in a different instance of the Server, this option fails with a "File in Use" error. This option also fails if the *expression2* does not specify a valid file.
	Not running	Creates a new instance of an Automation object specified by *expression1* that is associated with the file specified by the pathname in *expression2*. This option starts a new instance of the Server and instantiates the Automation object for the class that is initialized from the contents of the file. Often, the new Server, as well as the new Automation object, are invisibly created. This option fails if *expression2* does not specify a valid file.

Connection option	Server execution status	Connection behavior
4. CONNECT TO *expression2* WHERE *expression1* = ""	Running	Creates or connects to an Automation object that is associated with the file specified by the pathname in *expression2*. This option determines the identity of the Server (and hence the Automation object) from the file extension given in *expression2*. If more than one instance of the Server is running, this option randomly selects one (generally, the first one started). If the specified file is already open within the selected Server, this option connects to the Automation object that is instantiated for that file. If the file is not already open in the selected Server, this option opens the file and instantiates the specified Automation object for it. If the specified file is already open in a different instance of the Server, this option fails with a "File in Use" error. This option also fails if the *expression2* does not specify a valid file.
	Not running	Creates a new instance of an Automation object that is associated with the file specified by the pathname in *expression2*. This option determines the identity of the Server (and hence the Automation object) from the file extension given in *expression2*. This option starts a new instance of the Server and instantiates the Automation object for the class that is initialized from the contents of the file. Often, the new Server, as well as the new Automation object, are invisibly created. This option fails if *expression2* does not specify a valid file.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

The following procedure demonstrates several Automation object instantiations using the four basic connection options. It tries all of the options with the Microsoft® Excel Automation Server. Note that not all Automation Servers support all options. For example in Office 95, there is no Automation object for PowerPoint presentations. Thus, the file connection option (Option 3 in Table 20: Automation object connection options) does not work.

r-crea.p

```
/* Demonstration of connecting to an Automation Object in Excel using the
   different connection options. */

DEFINE VARIABLE curDir   AS CHARACTER  NO-UNDO.
DEFINE VARIABLE cEditor  AS CHARACTER  NO-UNDO
  VIEW-AS EDITOR SIZE 63 BY 1 LABEL "Result:" FONT 2.
DEFINE VARIABLE wordAppl AS COM-HANDLE NO-UNDO.
```

```
DEFINE BUTTON bExit
  LABEL "Exit" SIZE 16 BY 1.25 AUTO-GO.
DEFINE BUTTON bStart
  LABEL "Option 1 - Start Excel" SIZE 32 BY 1.25 .
DEFINE BUTTON bConnect
  LABEL "Option 2 - Connect to Active" SIZE 32 BY 1.25.
DEFINE BUTTON bConPerFile
  LABEL "Option 3 - Connect per File" SIZE 32 BY 1.25.
DEFINE BUTTON bConnectMon
  LABEL "Option 4 - Connect by Extension" SIZE 32 BY 1.25.

ASSIGN
  FILE-INFO:FILE-NAME = "."
  curDir              = FILE-INFO:FULL-PATHNAME.

FORM cEditor SKIP(0.5) bStart SPACE bConnect SPACE bConPerFile SPACE
  bConnectMon SKIP(0.5) bExit
  WITH FRAME a VIEW-AS DIALOG-BOX THREE-D FONT 6.
FRAME a:TITLE = "Testing CREATE Automation Object Statement".

ENABLE ALL WITH FRAME a.

ON CHOOSE OF bStart IN FRAME a DO:
  /* Option 1: CREATE expression1 Com-Handle-Var. */
  DEFINE VARIABLE excelAppl AS COM-HANDLE NO-UNDO.

  CREATE "Excel.Application" excelAppl.
  excelAppl:Visible = TRUE.
  excelAppl:Workbooks:Add.
  excelAppl:Range("A1"):Value = "testing CREATE".
  ASSIGN cEditor:SCREEN-VALUE = STRING(excelAppl:Range("A1"):Value).
  RELEASE OBJECT excelAppl.
END.

ON CHOOSE OF bConnect IN FRAME a DO:
/* Option 2: CREATE expression1 Com-Handle-Var CONNECT. */
  DEFINE VARIABLE excelAppl AS COM-HANDLE NO-UNDO.

  CREATE "Excel.Application" excelAppl CONNECT.
  excelAppl:Range("A2"):Value = "testing CONNECT".
  MESSAGE "Click me to continue!" VIEW-AS ALERT-BOX.
  ASSIGN cEditor:SCREEN-VALUE = STRING(excelAppl:Range("A2"):Value).
  excelAppl:Workbooks:Item(1):SaveAs(curDir + "\yyy.xls").
  excelAppl:Quit().
  RELEASE OBJECT excelAppl.
END.

ON CHOOSE OF bConPerFile IN FRAME a DO:
/* Option 3: CREATE expression1 Com-Handle-Var CONNECT TO expression2. */
  DEFINE VARIABLE excelAppl AS COM-HANDLE NO-UNDO.
  DEFINE VARIABLE cFileName AS CHARACTER  NO-UNDO
    INITIAL "\WorkSheets\Xplan.xls".

  CREATE "Excel.Sheet" excelAppl CONNECT TO cFileName.
  excelAppl:Visible = TRUE.
  excelAppl:Workbooks:Add.
  excelAppl:Range("A3"):Value = "testing CONNECT TO".
  ASSIGN cEditor:SCREEN-VALUE = STRING(excelAppl:Range("A3"):Value).
  RELEASE OBJECT excelAppl.
END.

ON CHOOSE OF bConnectMon IN FRAME a DO:
/* Option 4: CREATE "" Com-Handle-Var CONNECT TO expression2. */
  DEFINE VARIABLE excelAppl AS COM-HANDLE NO-UNDO.
  DEFINE VARIABLE cFileName AS CHARACTER  NO-UNDO
    INITIAL "\WorkSheets\Xplan.xls".

  CREATE "" excelAppl CONNECT TO cFileName.
```

```
  excelAppl:Range("A4"):Value = "testing CONNECT TO where expression1 = ''".
  MESSAGE "Click me to continue!" VIEW-AS ALERT-BOX.
  ASSIGN cEditor:SCREEN-VALUE = STRING(excelAppl:Range("A4"):Value).
  excelAppl:Workbooks:Item(1):SaveAs(curDir + "\zzz.xls").
  excelAppl:Quit().
  RELEASE OBJECT excelAppl.
END.
```

Notes

- You must ensure that any third-party Automation objects you want to instantiate are installed and correctly listed in the registry. For information on what Automation objects you can instantiate, see the documentation for the third-party product. Generally, these are the same Automation objects instantiated by the Visual Basic CreateObject and GetObject functions. You might also be able to view these Automation objects using the OpenEdge COM Object Viewer tool. For more information, see *OpenEdge Programming Interfaces*.
- The instantiation of an Automation object depends on the implementation of the Automation Server itself. Any Server registered for multiple use (REGCLS_MULTIPLE_USE flag) launches a single instance of the Server that handles multiple Automation object instantiation requests. Any Server registered single use (REGCLS_SINGLE_USE flag) launches a new instance of the Server for each instantiated Automation object.
- The four connection options in Table 20: Automation object connection options compare to the following Visual Basic function calls:
  - **Option 1** — CreateObject (*class*) or GetObject ("", *class*)
  - **Option 2** — GetObject (, *class*)
  - **Option 3** — GetObject (*pathname*, *class*)
  - **Option 4** — GetObject (*pathname*)
- Once you create or connect to an Automation object, you can reference its properties and methods.

See also

NO-ERROR option, RELEASE OBJECT statement

CREATE BROWSE statement
(Graphical interfaces only)

Creates a dynamic read-only browse widget or a dynamic updatable browse widget. Add browse columns to the widget with the ADD-LIKE-COLUMN, ADD-COLUMNS-FROM, and ADD-CALC-COLUMN methods. Specify the query for the widget with the QUERY attribute.

The dynamic updatable browse can only be a NO-ASSIGN browse—you must make all data assignments to the database.

Note: Does not apply to SpeedScript programming.

Syntax

```
CREATE BROWSE handle
[ IN WIDGET-POOL widget-pool-name ]
[ ASSIGN { attribute=expression } ... ]
[ trigger-phrase ]
```

handle

A variable of type HANDLE that the AVM sets for the new widget.

IN WIDGET-POOL *widget-pool-name*

Specifies the widget pool in which the object is created. If you do not specify a widget pool, the object is created in the current default widget pool. The browse will go away when its widget pool goes away, or when you delete it with the DELETE OBJECT statement.

ASSIGN { *attribute* = *expression* } . . .

Assigns specified values to attributes of the object. The *attribute* parameter must be the name of a valid attribute for the object and *expression* must evaluate to a valid value for that attribute.

trigger-phrase

A trigger phrase associated with the object. For more information, see the Trigger phrase reference entry.

Example

The following example creates a dynamic browse and adds columns to it:

r-dynbrws.p

```
/* r-dynbrws */
DEFINE VARIABLE name-hdl       AS HANDLE  NO-UNDO.
DEFINE VARIABLE num-hdl        AS HANDLE  NO-UNDO.
DEFINE VARIABLE address-hdl    AS HANDLE  NO-UNDO.
DEFINE VARIABLE calc-col-hdl   AS HANDLE  NO-UNDO.
DEFINE VARIABLE browse-hdl     AS HANDLE  NO-UNDO.
DEFINE VARIABLE buff-field-hdl AS HANDLE  NO-UNDO.
DEFINE VARIABLE brws-col-hdl   AS HANDLE  NO-UNDO.
DEFINE VARIABLE jx             AS INTEGER NO-UNDO.

DEFINE BUTTON btn-delete LABEL "Delete".
DEFINE BUTTON btn-quit   LABEL "&Quit"  AUTO-ENDKEY.

DEFINE FRAME MyFrame SKIP(10)
  btn-delete btn-quit
  WITH SIZE 80 BY 22.

DEFINE QUERY q1 FOR Customer SCROLLING.
OPEN QUERY q1 FOR EACH Customer NO-LOCK.

CREATE BROWSE browse-hdl
  ASSIGN
    TITLE     = "Dynamic Browse"
    FRAME     = FRAME MyFrame:HANDLE
    QUERY     = QUERY q1:HANDLE
```

```
    X         = 2
    Y         = 2
    WIDTH     = 74
    DOWN      = 10
    VISIBLE   = YES
    READ-ONLY = NO
    SENSITIVE = TRUE.

ON ROW-DISPLAY OF browse-hdl DO:
  IF VALID-HANDLE(calc-col-hdl) THEN
    calc-col-hdl:SCREEN-VALUE =
      STRING(Customer.CreditLimit - Customer.Balance).
END.

ASSIGN
  num-hdl      = browse-hdl:ADD-LIKE-COLUMN("Customer.CustNum").
  name-hdl     = browse-hdl:ADD-LIKE-COLUMN("Customer.Name").
  address-hdl  = browse-hdl:ADD-LIKE-COLUMN("Customer.Address").
  calc-col-hdl = browse-hdl:ADD-CALC-COLUMN("INT","->,>>>,>>9.99","",
    "Credit Left").

/* Refresh needs to be done if ADD-CALC-COLUMN is done after the browse is
   displayed. In ROW-DISPLAY trigger, we can only set the calc field's
   screen-value if the handle is set. And the handle is set after the
   ADD-CALC-COLUMN method is done. */

browse-hdl:refresh().
browse-hdl:EXPANDABLE = YES.

ON ROW-LEAVE OF browse-hdl DO:
  IF browse-hdl:CURRENT-ROW-MODIFIED THEN DO:
    REPEAT jx = 1 TO browse-hdl:NUM-COLUMNS:
      brws-col-hdl = browse-hdl:GET-BROWSE-COLUMN(jx).
      IF brws-col-hdl:MODIFIED THEN DO:
        buff-field-hdl = brws-col-hdl:BUFFER-FIELD.
        /* If buff-field-hdl is unknown, this is a calculated field and cannot
           be updated */
        IF buff-field-hdl NE ? THEN
          buff-field-hdl:BUFFER-VALUE = brws-col-hdl:SCREEN-VALUE.
      END.
    END.
  END.
END.

ON CHOOSE OF btn-delete DO:   /*  LABEL "DeleteDynBrowse". */
  DELETE WIDGET browse-hdl.
END.

ON CHOOSE OF btn-quit DO:
  QUIT.
END.

ENABLE ALL WITH FRAME MyFrame.
WAIT-FOR CLOSE OF CURRENT-WINDOW.
```

Notes

- If the browse's height is set using the DOWN attribute and a browse column is added, the browse's height may change to ensure that the number of DOWN is preserved. This may be due to the addition of the horizontal scrollbar or the growth of the column header.
- If the browse's height is set using the HEIGHT-CHARS or HEIGHT-PIXELS attributes, or through direct manipulation, and a browse column is added, the DOWN attribute may change to ensure that the specified height is preserved. This may be due to the addition of the horizontal scrollbar or the growth of the column header.

- The DISPLAY . . . WITH BROWSE *browse-name* statement cannot be used with a dynamic browse. Instead, the user must set the SCREEN-VALUE attributes.
- A dynamic browse's validation expression is restricted. It may not contain a CAN-FIND function. To reference the field, the FRAME-VALUE function must be used. The CAN-FIND function will still work for a static browse column.
- If a buffer-field is associated with a dynamic browse column, set the buffer-field's VALIDATE-EXPRESSION attribute before the dynamic browse column is added to the browse (via the ADD-LIKE-COLUMN( ) method). The validation expression is compiled at this time. If the VALIDATE-EXPRESSION attribute is changed later, it is ignored.
- You can use the ASSIGN option to assign a widget ID value to the WIDGET-ID attribute for this object. If you have enabled application-defined widget IDs in your ABL GUI application, by specifying the Use Widget ID (`-usewidgetid`) startup parameter, then the AVM uses this widget ID when it creates the widget at run time, instead of using the widget ID it normally generates by default. If you have not enabled application-defined widget IDs, then the AVM ignores this option setting at run time.

  For more information about the WIDGET-ID attribute, see its reference entry in the Handle Attributes and Methods Reference. For more information about the Use Widget ID (`-usewidgetid`) startup parameter, see *Startup Command and Parameter Reference*.

See also

ADD-CALC-COLUMN( ) method, ADD-COLUMNS-FROM( ) method, ADD-LIKE-COLUMN( ) method, CREATE QUERY statement, CREATE widget statement, DEFINE BROWSE statement, DEFINE QUERY statement, GET-BROWSE-COLUMN( ) method, QUERY attribute

CREATE BUFFER statement

Creates a dynamic buffer object.

Syntax

```
CREATE BUFFER handle FOR TABLE { table-name | table-handle | buffer-handle }
  [ BUFFER-NAME buffer-name ]
  [ IN WIDGET-POOL widget-pool-name ]
```

handle

A variable of type HANDLE that represents the handle of the buffer object.

FOR TABLE *table-name* | *table-handle* | *buffer-handle*

A character expression (*table-name*) that evaluates to a unique database table name or static temp-table name, a temp-table handle (*table-handle*), or to an existing buffer object handle (*buffer-handle*), each of which can specify the record source for which to create the buffer object.

If *table-name* is ambiguous, you must qualify the database table name with a database name or rename the temp-table. Otherwise, if the database table exists in multiple connected databases, the AVM creates the buffer in the first connected database.

BUFFER-NAME *buffer-name*

An expression of type CHARACTER that evaluates, at run time, to the name of the dynamic buffer you are creating. This option lets a dynamic query have multiple buffers for the same table.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates, at run time, to the name of the widget pool that contains the dynamic buffer.

Note: Widget pool names are not case-sensitive.

Example

The following example runs the query "for each customer" dynamically against the Sports2020 database using a purely dynamic buffer with no compile time references at all:

r-crtbuf.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.
DEFINE VARIABLE qh AS HANDLE  NO-UNDO.
DEFINE VARIABLE bh AS HANDLE  NO-UNDO.
DEFINE VARIABLE fh AS HANDLE  NO-UNDO EXTENT 10.

CREATE BUFFER bh FOR TABLE "Customer".
CREATE QUERY qh.

qh:SET-BUFFERS(bh).
qh:QUERY-PREPARE("FOR EACH Customer").
qh:QUERY-OPEN().
qh:GET-FIRST().

DISPLAY bh:NAME.

REPEAT ix = 1 TO 10:
  fh[ix] = bh:BUFFER-FIELD(ix).
  DISPLAY fh[ix]:NAME STRING(fh[ix]:BUFFER-VALUE).
END.

qh:QUERY-CLOSE().
bh:BUFFER-RELEASE().
DELETE OBJECT bh.
DELETE OBJECT qh.
```

The following code fragment shows several different ways you can create an alternate dynamic buffer for a static temp-table. Note the notation BUFFER tt1:HANDLE and TEMP-TABLE tt1:HANDLE for accessing the default buffer object handle and temp-table object handle, respectively, for the static temp-table, tt1:

```
DEFINE VARIABLE hbuf  AS HANDLE NO-UNDO. /* Default buffer object */
DEFINE VARIABLE htab  AS HANDLE NO-UNDO. /* Temp-table object */
DEFINE VARIABLE hbuf1 AS HANDLE NO-UNDO. /* 2nd non-default buffer object */

DEFINE TEMP-TABLE tt1 NO-UNDO
  FIELD x AS CHARACTER.

hbuf = BUFFER tt1:HANDLE.     /* Static temp-table's default buffer */
htab = TEMP-TABLE tt1:HANDLE. /* Static temp-table handle (not buffer) */

/* Different ways to create an alternate temp-table buffer */
CREATE BUFFER hbuf1
  FOR TABLE BUFFER tt1:HANDLE.       /* From static tt's default buffer */
CREATE BUFFER hbuf1 FOR TABLE hbuf.  /* From static tt's default buffer */
CREATE BUFFER hbuf1
  FOR TABLE TEMP-TABLE tt1:HANDLE.   /* From static tt's handle */
CREATE BUFFER hbuf1 FOR TABLE htab.  /* From static tt's handle */
CREATE BUFFER hbuf1 FOR TABLE "tt1". /* From static tt's name */
```

Notes

- If the character expression, *table-name*, identifies a temp-table defined as REFERENCE-ONLY, the statement sets *handle* to an unbound object that cannot function as a buffer object. To create a valid buffer object for such a table, use FOR TABLE *table-handle* or *buffer-handle* instead.
- Unless you need to use an alternate buffer, the most economical and cleanest way to obtain a buffer object handle for a table is to retrieve the handle for its default buffer. For example:

```
DEFINE VARIABLE hbuf AS HANDLE NO-UNDO.
DEFINE VARIABLE htab AS HANDLE NO-UNDO.

DEFINE TEMP-TABLE tt2 NO-UNDO
  FIELD x AS CHARACTER.

CREATE TEMP-TABLE htab.
htab:TEMP-TABLE-PREPARE( "dynTT" ).

/* Obtaining the default buffer for a table */
hbuf = BUFFER Customer:HANDLE.     /* For a database table */
hbuf = BUFFER tt2:HANDLE.          /* For a static temp-table */
hbuf = htab:DEFAULT-BUFFER-HANDLE. /* For a dynamic temp-table */
```

See also

CREATE QUERY statement, DEFINE BUFFER statement

CREATE CALL statement

Creates a call object, then stores a handle to it in the handle variable specified. The call object, its attributes, and its methods, are used by applications to invoke logic dynamically.

Syntax

```
CREATE CALL handle [ IN WIDGET-POOL widget-pool] [ NO-ERROR ]
```

handle

A HANDLE expression that indicates the name of a HANDLE variable into which a handle to the new call object is stored.

IN WIDGET-POOL *widget-pool*

A CHARACTER expression that indicates the name of the widget pool to contain the new call object.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Notes

- Unlike most ABL objects, the call object, by default, is assigned not to the closest unnamed widget pool, but rather to the SESSION widget pool.
- A call object is deleted automatically when its widget pool is deleted. To delete it earlier than its widget pool, use the DELETE OBJECT statement.

See also

Call object handle, DELETE OBJECT statement, NO-ERROR option

CREATE CLIENT-PRINCIPAL statement

Creates an instance of a client-principal object at run time in the INITIAL state (see the LOGIN-STATE attribute entry). Each client-principal instance is a security token that contains user credentials specific to one user login session. This login session can be used to set the user identity for one or more ABL sessions, one or more database connections, or an entire ABL application. This user identity must be defined in a security domain that is registered in a trusted domain registry that is used to authenticate or validate that identity.

Note: You use a client-principal object with the SET-CLIENT( ) method or SET-DB-CLIENT function to set the user identity for an ABL session or OpenEdge database. You can set only one user identity at any one point in time for each ABL session or database connection. However, you can maintain multiple client-principal instances for any number of user identities that you need to set and maintain during an application.

Syntax

```
CREATE CLIENT-PRINCIPAL client-principal-handle
  [ IN WIDGET-POOL widget-pool-name]
```

client-principal-handle

A variable of type HANDLE that contains the handle to the client-principal object.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates, at run time, to the name of the widget pool in which the client-principal is created.

Note: Widget pool names are not case sensitive.

- To use a client-principal object in the INITIAL state, you must set the required attributes to store the user's credentials and provide access to a domain registry that you trust to authenticate or validate the identity it represents for the ABL session or a database connection:
  - The local domain registry for a database connection is automatically loaded by OpenEdge and available to verify any identity assigned to the connection. Using OpenEdge data administration tools, you can also configure the OpenEdge database to ignore its local domain registry after the initial connection is established, and use the ABL session domain registry to securely set its connection identity. This allows multiple database connections to share a single domain registry to validate and set their connection identities
  - You can use the LOAD-DOMAINS( ) method on the SECURITY-POLICY system handle to load the ABL session domain registry from the local domain registry of a connected OpenEdge database. This automatically prevents the registration or loading of domains from outside the this database, but enables all database connections to share the same registry built from a single connected database. This method creates a session domain registry that can be used to authenticate or validate identity for both multi-tenant and non-multi-tenant database connections.
  - If you have not already loaded the ABL session registry using LOAD-DOMAINS( ), you can use the REGISTER-DOMAIN( ) method on the SECURITY-POLICY system handle to register and load domains individually into the session domain registry. You can then prevent additional domains from being loaded into the session registry using the LOCK-REGISTRATION( ) method on the SECURITY-POLICY system handle. An ABL session domain registry loaded and locked using REGISTER-DOMAIN( ) and LOCK-REGISTRATION( ) does not allow you to authenticate or validate identity for multi-tenant database connections. These methods only allow you to build a session domain registry that can be shared among non-multi-tenant database connections.
- If you do not specify a widget pool name, the object is created in the Session unnamed widget pool (not in the closest unnamed widget pool). The object goes away when its widget pool goes away or when you delete it using the DELETE OBJECT statement.

See also

Client-principal object handle, LOAD-DOMAINS( ) method, LOCK-REGISTRATION( ) method, REGISTER-DOMAIN( ) method, SET-CLIENT( ) method, SET-DB-CLIENT function

CREATE DATABASE statement

Creates a new OpenEdge database.

Syntax

```
CREATE DATABASE new-database [ FROM old-database [ NEW-INSTANCE ] ]
  [ REPLACE ] [ NO-ERROR ]
```

new-database

A CHARACTER expression that returns the absolute or relative pathname of the database you want to create. Any relative pathname is relative to the current working directory. If the database already exists, a new database is not created unless you specify REPLACE. The pathname is restricted to alphanumeric characters. Diacritical marks and the symbols \ " ' * ; | ? [ ] ( ) ! { } < > @ + = : ~ are not permitted.

FROM *old-database*

A CHARACTER expression that returns the name of the database whose schema and data you want to copy to the new database. The value of *old-database* can be an absolute or relative pathname or one of the special strings "EMPTY", "DEMO", or "Sports2020". If you omit this option, the AVM creates an empty database. Any relative pathname is relative to the current working directory.

NEW-INSTANCE

If specified, the AVM assigns the new database a new globally unique identifier (GUID) value as the database identifier. If not specified, the AVM assigns the new database the same GUID database identifier as the old database.

When you create a new database by copying an existing database provided by OpenEdge (such as the empty database, demo database, or Sports2020 database), the AVM always assigns the new database a new GUID database identifier.

Note: Use this option only when creating a new database by copying an existing database.

REPLACE

If specified and a database already exists with the name specified by *new-database*, the existing database is deleted and replaced with the new database. If not specified and a database already exists with the name specified by *new-database*, an error occurs.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

This procedure prompts for the name of a database to connect. If the database does not exist, the procedure creates it.

r-credb.p

```
DEFINE VARIABLE dbname AS CHARACTER NO-UNDO LABEL "Database" FORMAT "x(65)".

/* Prompt the user for the name of a demo database to connect. */
SET dbname HELP "Enter the name of your database."
  WITH FRAME dbname-frame SIDE-LABELS.

/* If the entered name does not have the .db suffix, add it. This is necessary
   for the search function to work correctly. */
IF LENGTH(dbname) < 3 THEN
  dbname = dbname + ".db".
ELSE IF SUBSTRING(dbname, LENGTH(dbname) - 2) = ".db" THEN
  dbname = dbname + ".db".

/* If the database does not exist, create it from Sports2020. */
IF SEARCH(dbname) = ? THEN DO:
  MESSAGE "Database does not exist. Do you want to create it?"
    VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO TITLE "Connect Database"
    UPDATE create-it AS LOGICAL.

  IF create-it THEN DO:
    CREATE DATABASE dbname FROM "Sports2020".
    MESSAGE "New database created:" dbname.
  END.
  ELSE UNDO, RETRY.
END.

/* Connect the database. */
CONNECT VALUE(dbname) -1.
```

Notes

If you omit the FROM option, the AVM uses the empty database.

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DELETE ALIAS statement, DISCONNECT statement, ERROR-STATUS system handle, FRAME-DB function, LDBNAME function, NO-ERROR option, NUM-DBS function, PDBNAME function, SDBNAME function

CREATE DATASET statement

Creates a ProDataSet object dynamically at run time. The ProDataSet object that is created is empty.

Note: You can use the ADD-BUFFER( ) method and SET-BUFFERS( ) method to add buffers to a dynamic ProDataSet object.

Syntax

```
CREATE DATASET dataset-handle [ IN WIDGET-POOL widget-pool-name ]
```

dataset-handle

A variable of type HANDLE that represents the handle of the dynamic ProDataSet object.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates, at run time, to the name of the widget pool in which the dynamic ProDataSet object is created.

Note: Widget pool names are not case-sensitive.

Notes

- If you do not specify a widget pool name, the object is created in the Session unnamed widget pool (not in the closest unnamed widget pool). The object goes away when its widget pool goes away or when you delete it using the DELETE OBJECT statement.
- If the ProDataSet object serves as an OUTPUT parameter and you specify a widget pool, the widget pool must outlive the called procedure.

Example

Following is an example of how to create a dynamic ProDataSet object for Orders and their Orderlines:

```
DEFINE VARIABLE hDset AS HANDLE NO-UNDO.
DEFINE VARIABLE hRel  AS HANDLE NO-UNDO.

CREATE DATASET hDset.

hDset:SET-BUFFERS(BUFFER ttOrder:HANDLE, BUFFER ttOrderLine:HANDLE).

hRel = hDset:ADD-RELATION(BUFFER ttOrder:HANDLE, BUFFER ttOrderLine:HANDLE,
  "OrderNum,OrderNum").
```

See also

ProDataSet object handle, DEFINE DATASET statement

CREATE DATA-SOURCE statement

Creates a data-source object dynamically at run time.

Syntax

```
CREATE DATA-SOURCE data-source-handle [ IN WIDGET-POOL widget-pool-name ]
```

data-source-handle

A variable of type HANDLE that represents the handle of the dynamic data-source object.

`IN WIDGET-POOL` *`widget-pool-name`*

An expression of type CHARACTER that evaluates, at run time, to the name of the widget pool in which the dynamic data-source object is created.

Note: Widget pool names are not case-sensitive.

Notes

If you do not specify a widget pool name, the dynamic data-source object is created in the closest unnamed widget-pool, by default. The object goes away when its widget pool goes away or when you delete it using the DELETE OBJECT statement.

See also

Data-source object handle, DEFINE DATA-SOURCE statement

CREATE QUERY statement

Creates a dynamic query.

Syntax

```
CREATE QUERY handle
  [ IN WIDGET-POOL widget-pool-name ]
```

`handle`

A variable of type HANDLE that represents the handle of the query object.

`IN WIDGET-POOL` *`widget-pool-name`*

An expression of type CHARACTER that evaluates, at run time, to the name of the widget pool that contains the dynamic query.

Note: Widget pool names are not case-sensitive.

Example

The following example creates a dynamic query with a static buffer and a dynamic predicate (WHERE clause) which is resolved at run time:

r-crtqry.p

```
/* r-crtqry.p */

DEFINE VARIABLE hQuery  AS HANDLE  NO-UNDO.
DEFINE VARIABLE iNumVar AS INTEGER NO-UNDO INITIAL 10.

CREATE QUERY hQuery.

hQuery:SET-BUFFERS(BUFFER Customer:HANDLE).
hQuery:QUERY-PREPARE("FOR EACH Customer WHERE Customer.CustNum < " +
  STRING(iNumVar)).
hQuery:QUERY-OPEN.

REPEAT WITH FRAME y:
  hQuery:GET-NEXT().
  IF hQuery:QUERY-OFF-END THEN LEAVE.
  DISPLAY
    Customer.CustNum
    Customer.Name FORMAT "x(30)"
    Customer.City FORMAT "X(20)".
END.

hQuery:QUERY-CLOSE().
DELETE OBJECT hQuery.
```

Note

- CREATE-QUERY must be followed by the QUERY-PREPARE( ) and QUERY-OPEN() methods before the query can be run.

See also

CREATE BUFFER statement, DEFINE QUERY statement, QUERY-OPEN( ) method, QUERY-PREPARE( ) method

CREATE SAX-ATTRIBUTES statement

Creates an instance of a SAX-attributes object and assigns its handle to the handle variable specified. Use this object to access and manage the attribute list for an XML element either being read or written with the SAX-reader or SAX-writer object.

Syntax

```
CREATE SAX-ATTRIBUTES handle [ IN WIDGET-POOL pool-name ] [ NO-ERROR ]
```

handle

Variable of type HANDLE which stores the handle of the new SAX-attributes object.

IN WIDGET-POOL *pool-name*

Specifies the widget pool where the AVM creates the new object. If you do not specify a widget pool, the AVM creates the object in the current default widget pool.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

In ABL, a SAX-attributes object can be automatically created and managed by the SAX parser, or programatically created and managed by you. Automatic SAX-attributes objects are created, populated, and destroyed during the scope of the startElement callback procedure. The startElement callback is called by the SAX-reader object each time the SAX parser encounters new XML element during a read operation. The SAX parser populates the SAX-attributes object, but you may interact with the object during the scope of the startElement callback procedure. For example, you may add elements, update elements, and remove elements from the SAX-attributes object.

At the conclusion of the callback, the SAX-attributes object is deleted and your access to the element list is lost. To save the XML attributes list, you would need to create another SAX-attributes object that is not tied to the scope of the startElement callback procedure. Then, from within the startElement callback, you would use the COPY-SAX-ATTRIBUTES( ) method on the new object and provide a handle to the automatically created SAX-attributes object.

Finally, you would likely pass the XML element and its attributes to a SAX-writer object to concurrently build a new XML document while reading the source XML document.

Example

The following code snippets assembled form one complete example of a concurrent XML read and write operation. The first snippet shows the main block of code:

```
DEFINE VARIABLE cFileRead  AS CHARACTER NO-UNDO INITIAL "custorders.xml".
DEFINE VARIABLE cFileWrite AS CHARACTER NO-UNDO INITIAL "custitems.xml".

DEFINE VARIABLE hReader AS HANDLE  NO-UNDO.
DEFINE VARIABLE hWriter AS HANDLE  NO-UNDO.
DEFINE VARIABLE hSaxAtt AS HANDLE  NO-UNDO.
DEFINE VARIABLE lok     AS LOGICAL NO-UNDO.

CREATE SAX-READER hReader.
CREATE SAX-WRITER hWriter.
CREATE SAX-ATTRIBUTES hSaxAtt.

/* Uses THIS-PROCEDURE as the SAX handler. All callbacks are below. */

lok = hReader:SET-INPUT-SOURCE("file", cFileRead).
lok = hWriter:SET-OUTPUT-DESTINATION("file", cFileWrite).

hReader:SAX-PARSE().
IF ERROR-STATUS:ERROR THEN DO:
  MESSAGE "Problem reading the XML file:" cFileRead.
  MESSAGE ERROR-STATUS:GET-MESSAGE(1).
END.
ELSE
  MESSAGE "XML file converted. Written to" cFileWrite.

DELETE OBJECT hReader.
DELETE OBJECT hWriter.
DELETE OBJECT hSaxAtt.
```

This snippet contains the callbacks for starting and ending a document read. Note that embedded in the steps of the read operation are the corresponding steps for a concurrent write operation:

```
/* SAX-reader callbacks below*/

/* Context controller */
DEFINE VARIABLE lWriteChar AS LOGICAL NO-UNDO.

PROCEDURE startdocument:
  ASSIGN
    hWriter:ENCODING = "utf-8"
    lok              = hWriter:START_DOCUMENT().
END PROCEDURE.

PROCEDURE enddocument:
  lok = hWriter:END-DOCUMENT().
END PROCEDURE.
```

The startElement callback does most of the processing. Depending on the values being read by SAX-reader, the callback decides what to write to the new XML document:

```
PROCEDURE startelement:
  DEFINE INPUT PARAMETER namespaceuri AS CHARACTER NO-UNDO.
  DEFINE INPUT PARAMETER localname    AS CHARACTER NO-UNDO.
  DEFINE INPUT PARAMETER qname        AS CHARACTER NO-UNDO.
  DEFINE INPUT PARAMETER attributes   AS HANDLE    NO-UNDO.

  CASE qname:
    WHEN "customer" THEN ASSIGN
      lWriteChar = FALSE
      lok        = hSaxAtt:COPY-SAX-ATTRIBUTES(attributes)
      lok        = hWriter:START-ELEMENT(qname).
    WHEN "name" THEN ASSIGN
      lWriteChar = TRUE
      lok        = hSaxAtt:REMOVE-ATTRIBUTE("id")
      lok        = hWriter:START-ELEMENT(qname, hSaxAtt).
    WHEN "order" THEN ASSIGN
      lWriteChar = FALSE
      lok        = hWriter:START-ELEMENT(qname).
    WHEN "item" THEN ASSIGN
      lWriteChar = FALSE
      lok        = hWriter:START-ELEMENT(qname)
      lok        = hWriter:WRITE-CHARACTERS
        (attributes:GET-VALUE-BY-QNAME("description"))
      lok        = hWriter:END-ELEMENT(qname).
    WHEN "custorders" THEN ASSIGN
      lWriteChar = FALSE
      lok        = hWriter:START-ELEMENT("custitems").
  END CASE.
END PROCEDURE.
```

The last snippet completes the set of callback procedures needed for this example:

```
PROCEDURE endelement:
  DEFINE INPUT PARAMETER namespaceuri AS CHARACTER NO-UNDO.
  DEFINE INPUT PARAMETER localname    AS CHARACTER NO-UNDO.
  DEFINE INPUT PARAMETER qname        AS CHARACTER NO-UNDO.

  IF qname = "customer" OR qname = "name" OR qname = "order" THEN
    lok = hWriter:END-ELEMENT(qname).

  IF qname = "custorders" THEN
    lok = hWriter:END-ELEMENT("custitems").
END PROCEDURE.

PROCEDURE characters:
  DEFINE INPUT PARAMETER chararray   AS LONGCHAR NO-UNDO.
  DEFINE INPUT PARAMETER arraylength AS INTEGER  NO-UNDO.

  IF lWriteChar THEN
    lok = hWriter:WRITE-CHARACTERS(chararray).
END PROCEDURE.
```

See also

NO-ERROR option, SAX-attributes object handle

CREATE SAX-READER statement

Creates an instance of a SAX-reader object and assigns its handle to the handle variable you specify. The SAX-reader object can be used to control the parsing of XML source.

Syntax

```
CREATE SAX-READER handle
  [ IN WIDGET-POOL pool-name ] [ NO-ERROR ]
```

handle

A variable of type HANDLE into which CREATE SAX-READER stores the new handle.

IN WIDGET-POOL *pool-name*

A CHARACTER expression indicating the widget pool in which the object is created. If you do not specify a widget pool, the object is created in the current default widget pool.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

See also

NO-ERROR option, SAX-reader object handle

CREATE SAX-WRITER statement

Creates an instance of a SAX-writer object and assigns its handle to the handle variable specified. Use this object to write an XML document using the SAX interface.

Syntax

```
CREATE SAX-WRITER handle [ IN WIDGET-POOL pool-name ] [ NO-ERROR ]
```

handle

Variable of type HANDLE which stores the handle of the new SAX-writer object.

IN WIDGET-POOL *pool-name*

Specifies the widget pool where the AVM creates the new object. If you do not specify a widget pool, the AVM creates the object in the current default widget pool.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

See also

NO-ERROR option, SAX-writer object handle

CREATE SERVER statement

Creates an instance of a server object and assigns its handle to the handle variable you specify.

Syntax

```
CREATE SERVER handle
  [ ASSIGN { attribute = expression } ... ]
```

handle

A variable of type HANDLE into which CREATE SERVER stores the new server handle.

ASSIGN {*attribute = expression*} . . .

Assigns specified values to attributes of the handle. The *attribute* parameter must be the name of a valid attribute for a server handle, and the *expression* parameter must evaluate to a valid value for the attribute.

Note

You can use a server handle as a connection point to an application server. For more information on server handles, see the Server object handle entry.

See also

DELETE OBJECT statement, RUN statement, Server object handle

CREATE SERVER-SOCKET statement

Creates an instance of a server socket object and assigns it to the handle variable specified. It is through this object that a socket-based server application can listen for connections on a TCP/IP port.

Note: Does not apply to SpeedScript programming.

Syntax

```
CREATE SERVER-SOCKET handle [ NO-ERROR ]
```

handle

Variable of type HANDLE into which the CREATE SERVER-SOCKET statement stores the new server socket handle.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Notes

- An application can only create one server socket object. This statement will raise ERROR if an application tries to create multiple objects.
- A server socket object cannot be used with a Progress Application Server (PAS) for OpenEdge instance.

See also

CREATE SOCKET statement, DELETE OBJECT statement, NO-ERROR option, Server socket object handle, Socket object handle

CREATE SOAP-HEADER statement

Creates an instance of a SOAP-header object dynamically at run time, and assigns its handle to the specified handle variable.

Syntax

```
CREATE SOAP-HEADER handle [ IN WIDGET-POOL widget-pool-name ]
```

handle

A variable of type HANDLE that represents the handle of the SOAP-header object.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates to the name of the widget pool in which the dynamic SOAP-header object is created.

Note: Widget pool names are not case-sensitive.

Notes

- The SOAP HEADER object passed to the response callback is implicitly created by the AVM. In order to pass a SOAP HEADER object back from the request callback, the application needs to explicitly create it or use an object that it has previously saved.
- The SOAP-header object is either implicitly created by the AVM or explicitly created by the application using the CREATE SOAP-HEADER statement. In either case, the application is responsible for deleting the object.

  You can delete a SOAP-header object and its underlying XML in one of two ways. You can:

  - Use the DELETE OBJECT statement to delete the SOAP-header object directly.
  - Set the `lDeleteOnDone` parameter in the request header callback procedure to TRUE, which directs the AVM to delete the SOAP header object after it copies the object's contents to the outbound SOAP message.

  For more information about deleting SOAP-header objects, see *Develop Web Services for OpenEdge*.

See also

SOAP-header object handle

CREATE SOAP-HEADER-ENTRYREF statement

Creates an instance of a SOAP-header-entryref object dynamically at run time, and assigns its handle to the specified handle variable.

Syntax

```
CREATE SOAP-HEADER-ENTRYREF hshEntry [ IN WIDGET-POOL widget-pool-name ]
```

hshEntry

A variable of type HANDLE that represents the handle of the SOAP-header-entryref object.

`IN WIDGET-POOL` *`widget-pool-name`*

An expression of type CHARACTER that evaluates to the name of the widget pool in which the dynamic SOAP-header-entryref object is created. A SOAP-header-entryref object is created in the closest unnamed widget-pool, by default.

Note: Widget pool names are not case-sensitive.

Notes

The SOAP-header-entryref object is explicitly created by the application using the CREATE SOAP-HEADER-ENTRYREF statement. The application is responsible for deleting this object. Use the DELETE OBJECT statement to delete a SOAP-header-entryref object without deleting its underlying XML. Use the DELETE-HEADER-ENTRY() method to delete the XML underlying the SOAP-header-entryref object without deleting the object.

See also

SOAP-header-entryref object handle

CREATE SOCKET statement

Creates a socket object and assigns it to the handle variable specified. It is through this object that the application can connect to a TCP/IP port and read and write on the socket bound to the port.

Syntax

```
CREATE SOCKET handle [ NO-ERROR ]
```

`handle`

Variable of type HANDLE into which CREATE SOCKET stores the new socket handle.

`NO-ERROR`

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

See also

CREATE SERVER-SOCKET statement, DELETE OBJECT statement, NO-ERROR option, Server socket object handle, Socket object handle

CREATE TEMP-TABLE statement

Creates a TEMP-TABLE dynamically at run time. The TEMP-TABLE that is created is empty and must be defined using ADD/CREATE methods.

Syntax

```
CREATE TEMP-TABLE handle
  [ IN WIDGET-POOL widget-pool-name ]
```

handle

A variable of type HANDLE that represents the handle of the TEMP-TABLE object.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates, at run time, to the name of the widget pool that contains the dynamic TEMP-TABLE.

Note: Widget pool names are not case-sensitive.

Note

You can collect run-time statistics about usage of temp-tables in an ABL session. For more information on collecting statistics for temp-table usage, see *Troubleshoot ABL Applications*.

Example

The following example creates a TEMP-TABLE like the `Order` table and populates it from the `Order` table. In addition, the corresponding `SalesRep` name is added from the `SalesRep` table, as shown:

r-cretmpt.p

```
DEFINE VARIABLE tth          AS HANDLE NO-UNDO.
DEFINE VARIABLE bh           AS HANDLE NO-UNDO.
DEFINE VARIABLE qh           AS HANDLE NO-UNDO.
DEFINE VARIABLE buf-ord-hndl AS HANDLE NO-UNDO.
DEFINE VARIABLE buf-rep-hndl AS HANDLE NO-UNDO.
DEFINE VARIABLE fld1         AS HANDLE NO-UNDO.
DEFINE VARIABLE fld2         AS HANDLE NO-UNDO.

/* Get database table handles */
buf-ord-hndl = BUFFER Order:HANDLE.
buf-rep-hndl = BUFFER SalesRep:HANDLE.

/* Create an empty, undefined TEMP-TABLE */
CREATE TEMP-TABLE tth.
/* Give it Order table's fields & indexes */
tth:CREATE-LIKE(buf-ord-hndl).
/* Add field like SalesRep.RepName */
tth:ADD-LIKE-FIELD("RepName","SalesRep.RepName").
/* No more fields will be added */
tth:TEMP-TABLE-PREPARE("ordx").

/* Get the buffer handle for the temp-table */
bh = tth:DEFAULT-BUFFER-HANDLE.

/* Populate the temp-table from order */
FOR EACH Order NO-LOCK:
  bh:BUFFER-CREATE.
  bh:BUFFER-COPY(buf-ord-hndl).
  /* Add the corresponding salesrep name */
  FIND SalesRep NO-LOCK WHERE SalesRep.SalesRep = Order.SalesRep NO-ERROR.
  IF AVAILABLE SalesRep THEN
```

```
    bh:BUFFER-COPY(buf-rep-hndl,?,"RepName,repname").
END.

/* Run a query to access the TEMP-TABLE */
CREATE QUERY qh.
qh:SET-BUFFERS(bh).
qh:QUERY-PREPARE("FOR EACH ordx WHERE ordx.OrderNum < 50 BY ordx.RepName").
qh:QUERY-OPEN().

fld1 = bh:BUFFER-FIELD("OrderNum").
fld2 = bh:BUFFER-FIELD("RepName").

/* Display the order number and the salesrep name */
REPEAT:
  qh:GET-NEXT().
  IF qh:QUERY-OFF-END THEN LEAVE.
  DISPLAY fld1:BUFFER-VALUE() FORMAT "X(10)".
  DISPLAY fld2:BUFFER-VALUE() FORMAT "X(20)".
END.

qh:QUERY-CLOSE().
bh:BUFFER-RELEASE().
DELETE OBJECT tth.
DELETE OBJECT qh.
```

Notes

- Once the TEMP-TABLE fields and indexes are defined using the ADD/CREATE methods, the definition must be terminated by using the TEMP-TABLE-PREPARE method before the TEMP-TABLE can be used.
- You can set the *before-table-exp* parameter to TRUE in the TEMP-TABLE-PREPARE method to explicitly create a before-table for a dynamic temp-table. See the TEMP-TABLE-PREPARE method entry for more information.
- Once the TEMP-TABLE is prepared, it can be manipulated by using its buffer object handle which is retrieved using the DEFAULT-BUFFER-HANDLE attribute. All the BUFFER methods are available to the dynamic TEMP-TABLE.
- The dynamic temp-table object is scoped like the buffer object. It is created in a widget pool and ends when the widget pool ends or when it is deleted with the DELETE OBJECT statement. You may not delete the default buffer object belonging to a dynamic temp-table.
- Errors for dynamic temp-tables do not automatically raise the ERROR condition since they occur inside a widget expression. All the methods that can have errors return FALSE if an error occurs, so they must be tested. If NO-ERROR is in effect in the statement containing the widget reference, no messages display, but they can be retrieved from the ERROR-STATUS system handle.

See also

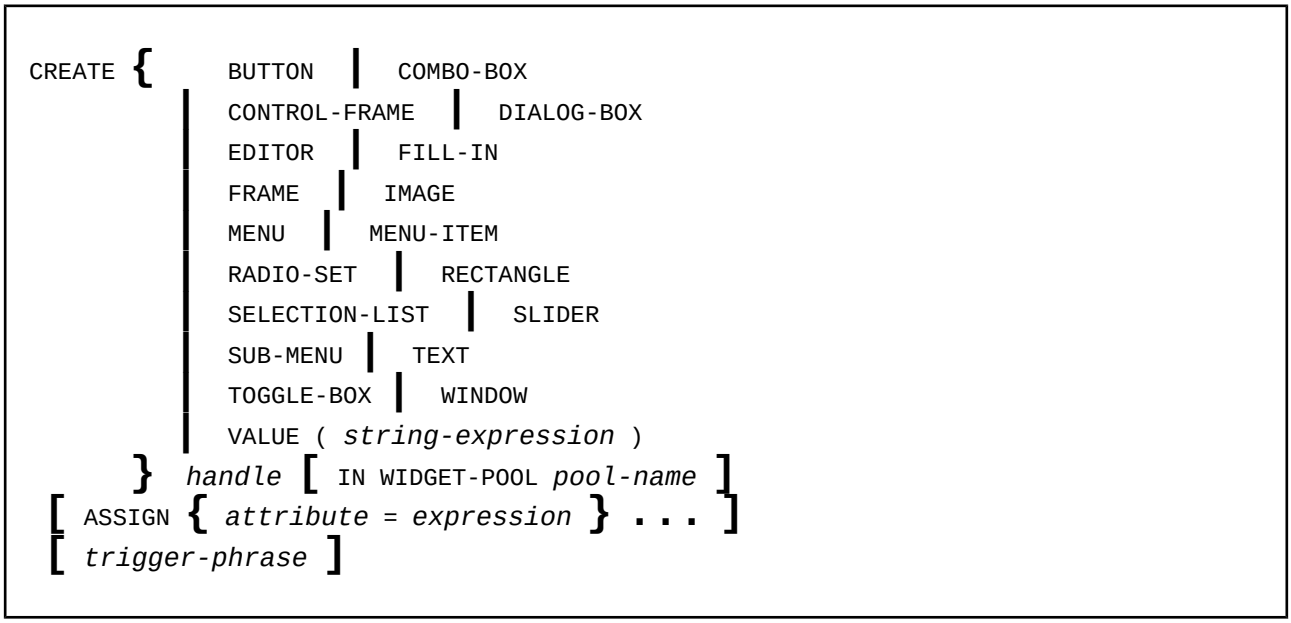
DEFINE TEMP-TABLE statement, TEMP-TABLE-PREPARE( ) method

CREATE widget statement

Creates a dynamic object, such as a widget object.

Note: Does not apply to SpeedScript programming.

Syntax



`VALUE ( string-expression )`

An expression of type CHARACTER that evaluates to the type of object you want to create (for example, BUTTON) with any combination of uppercase and lowercase characters.

handle

A variable of type HANDLE that the AVM sets for the new widget.

`IN WIDGET-POOL` *pool-name*

Specifies the widget pool in which the object is created. If you do not specify a widget pool, the object is created in the current default widget pool.

`ASSIGN {`*attribute* `=` *expression*`} ...`

Assigns specified values to attributes of the object. The *attribute* parameter must be the name of a valid attribute for the object and *expression* must evaluate to a valid value for that attribute.

trigger-phrase

A trigger phrase associated with the object. For more information, see the Trigger phrase reference entry.

Example

This procedure creates a dynamic button that displays a list of customer names:

r-dynbut.p

```
DEFINE VARIABLE but1 AS HANDLE NO-UNDO.

DISPLAY "Dynamic Button Example" SKIP(3) WITH FRAME x SIDE-LABELS.

OPEN QUERY all-custs FOR EACH Customer.
GET FIRST all-custs.
DISPLAY Customer.Name WITH FRAME x.

CREATE BUTTON but1 ASSIGN
  ROW       = 3
  COLUMN    = 5
  LABEL     = "Next Customer"
  FRAME     = FRAME x:HANDLE
  SENSITIVE = TRUE
  VISIBLE   = TRUE
  TRIGGERS:
    ON CHOOSE DO:
      GET NEXT all-custs.
      DISPLAY Customer.Name WITH FRAME x.
    END.
  END TRIGGERS.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- Attribute assignments you specify in the CREATE widget statement are processed in the order they appear. In some cases you must supply the attributes in proper order. For example, you cannot set the SENSITIVE or VISIBLE attributes for a field-level widget until you have set its FRAME attribute.
- If you are setting the FORMAT attribute and specifying an initial SCREEN-VALUE for the widget, assign the FORMAT before the SCREEN-VALUE. Otherwise, the default format is applied to the SCREEN-VALUE which might cause truncation or other formatting errors.
- You can use the ASSIGN option to assign a widget ID value to the WIDGET-ID attribute for this object. If you have enabled application-defined widget IDs in your ABL GUI application, by specifying the Use Widget ID (`-usewidgetid`) startup parameter, then the AVM uses this widget ID when it creates the widget at run time, instead of using the widget ID it normally generates by default. If you have not enabled application-defined widget IDs, then the AVM ignores this option setting at run time.

  For more information about the WIDGET-ID attribute, see its reference entry in the Handle Attributes and Methods Reference. For more information about the Use Widget ID (`-usewidgetid`) startup parameter, see *Startup Command and Parameter Reference*.
- If you create a frame to use as a DDE frame, you must realize the frame (set its VISIBLE attribute to TRUE) before using it as a conversation end-point. If you want the DDE frame to remain invisible during its use in a DDE conversation, set its HIDDEN attribute to TRUE after realizing the frame. For information on DDE frames, see *OpenEdge Programming Interfaces*.

See also

CREATE QUERY statement, CREATE WIDGET-POOL statement, DEFINE FRAME statement, DELETE WIDGET statement, DELETE WIDGET-POOL statement, Trigger phrase

CREATE WIDGET-POOL statement

Creates a named or unnamed widget pool in which to contain most dynamic widgets and other handle-based objects created during an ABL session.

Note: Does not apply to SpeedScript programming.

Syntax

```
CREATE WIDGET-POOL
  [ pool-name [ PERSISTENT ] ]
  [ NO-ERROR ]
```

pool-name

A character-string expression that specifies the name for a *named widget pool* you are creating. Widget pool names are not case sensitive.

If you omit this option, an unnamed widget pool is created and scoped to the procedure or class-based method. That is, a *routine-scoped unnamed widget pool* and its contents remain in effect as long as the procedure or method is on the call stack, and the pool and its contents are automatically deleted when the procedure or method is removed from the call stack.

PERSISTENT

Specifies that the named widget pool is persistent. This means that the pool and any widgets in it remain allocated after the current procedure or method terminates. If you do not specify this option, the pool and its contents are automatically deleted when procedure or method execution ends.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

The following example lets you create a series of dynamic buttons. All the buttons are created within a named widget pool. Because the widget pool is created within a trigger, it is defined as persistent so that it remains allocated after the trigger ends. You can at any time choose to delete the entire widget pool and start over.

r-widpl.p

```
DEFINE VARIABLE wh AS HANDLE NO-UNDO.

DEFINE BUTTON b_create LABEL "Create Button".
DEFINE BUTTON b_del    LABEL "Delete Buttons".
DEFINE BUTTON b_quit LABEL "Quit"
  TRIGGERS:
    ON CHOOSE DO:
      IF VALID-HANDLE(wh) THEN
        DELETE WIDGET-POOL "new-buttons".
      QUIT.
    END.
```

```
  END.

DEFINE FRAME butt-frame
  b_create b_del b_quit
  WITH ROW SCREEN-LINES - 2.

DEFINE FRAME new-buttons
  WITH SIZE 76 BY 11 CENTERED ROW 2 TITLE "New Buttons".

ON CHOOSE OF b_create IN FRAME butt-frame DO:
  STATUS INPUT "Press RETURN to select a new button".
  IF wh = ? OR NOT VALID-HANDLE(wh) THEN
    CREATE WIDGET-POOL "new-buttons" PERSISTENT.
  CREATE BUTTON wh IN WIDGET-POOL "new-buttons" ASSIGN
    FRAME     = FRAME new-buttons:HANDLE
    ROW       = RANDOM(2, 9)
    COLUMN    = RANDOM(2, 58)
    LABEL     = "BUTTON " + STRING(ETIME)
    SENSITIVE = TRUE
    VISIBLE   = TRUE
    TRIGGERS:
      ON CHOOSE PERSISTENT RUN dispmsg.
    END.
END.

ON CHOOSE OF b_del IN FRAME butt-frame DO:
  IF VALID-HANDLE(wh) THEN
    DELETE WIDGET-POOL "new-buttons".
  STATUS INPUT.
END.

ENABLE b_create b_del b_quit WITH FRAME butt-frame.

DO ON ENDKEY UNDO, LEAVE:
  WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame.
END.

IF VALID-HANDLE(wh) THEN
  DELETE WIDGET-POOL "new-buttons".

PROCEDURE dispmsg:
  MESSAGE "You chose button " SELF:LABEL.
END PROCEDURE.
```

Notes

- The AVM automatically creates a persistent unnamed widget pool (*session widget pool*) at the start of each session. Most applications use only this session widget pool.
- In general, unnamed widget pools cannot persist beyond the scope of the procedure or method in which they are created, except the session widget pool, which is created by the AVM.
- If you create an unnamed widget pool in the main block of a persistent procedure or you instantiate a class where the USE-WIDGET-POOL option is defined somewhere in its hierarchy, the AVM creates an *object-persistent unnamed widget pool* that persists for the lifetime of the persistent procedure or class-based object, respectively. This object-persistent widget pool then becomes the default widget pool for any internal procedure of the persistent procedure or any method of the instantiated class that is invoked from outside the respective persistent procedure or instantiated class. However, any routine-scoped unnamed widget pool created by these internal procedures or methods supersedes this object-persistent widget pool. For more information on the USE-WIDGET-POOL option, see the CLASS statement reference entry.
- When you create an unnamed widget pool, it automatically becomes the default widget pool. Each subsequent dynamically created widget is placed in this unnamed pool unless you specifically assign it to another pool. This unnamed pool remains the default widget pool until it is deleted or you create another unnamed widget

pool. Thus, if you create no unnamed widget pools, all dynamically created widgets go into the session widget pool, unless assigned to another pool. If you create any additional unnamed widget pools, either object-persistent or routine-scoped, the most locally scoped and recently created unnamed widget pool becomes the default widget pool for all subsequently created dynamic widgets.

- Persistent widget pools remain allocated until they are explicitly deleted (with the DELETE WIDGET-POOL statement) or until the end of the ABL session that created them.
- All named widget pools are globally scoped. While a named widget pool is allocated, any procedure within the same process can access that widget pool. The name of a widget pool must be unique among all widget pools for the process. If you try to create a widget pool with the same name as an existing pool, the AVM raises the ERROR condition.
- If a recursive procedure or method creates an unnamed widget pool, each iteration of that procedure or method creates a separate pool. If a recursive routine creates a named widget pool, you must ensure that only one iteration creates the pool (where all iterations can share it) or use a different name in each iteration (where each creates and uses its own pool).
- You might want to create a new, unnamed widget pool just before invoking a new procedure or method and then delete that pool when the procedure or method returns. This ensures that any dynamic widgets created by that procedure or method in the default pool are deleted immediately. For example:

```
CREATE WIDGET-POOL.
RUN xyz.p.
DELETE WIDGET-POOL.
```

Similarly, you might want to store all dynamic widgets for a subsystem within a specific named pool. For example:

```
CREATE WIDGET-POOL "oe-pool".
RUN ord-ent.p
DELETE WIDGET-POOL "oe-pool".
```

In this example, the procedure ord-ent.p must reference the oe-pool for each dynamic widget it creates.

See also

CLASS statement, CREATE widget statement, DELETE WIDGET-POOL statement, NO-ERROR option

CREATE X-DOCUMENT statement

Creates a handle for an XML document object. To use the XML document, you must add new nodes using the CREATE-NODE( ) method, the CREATE-NODE-NAMESPACE( ) method, or populate the document from an existing file using the LOAD( ) method.

Note: To ensure consistency across all nodes in an XML document, use either the CREATE-NODE-NAMESPACE( ) method or the CREATE-NODE( ) method to build an XML document; do not use both methods within a single document.

Syntax

```
CREATE X-DOCUMENT handle [ IN WIDGET-POOL widget-pool-name ]
```

handle

A variable of type HANDLE into which CREATE X-DOCUMENT stores the new handle.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates, at run time, to the name of the widget pool that contains the XML document object.

Note: Widget pool names are not case-sensitive.

Example

The following code fragment depicts creating an XML document object:

```
DEFINE VARIABLE hXdoc AS HANDLE NO-UNDO.
. . .
CREATE X-DOCUMENT hXdoc.
. . .
```

See also

CREATE X-NODEREF statement, DELETE OBJECT statement, X-document object handle, X-noderef object handle

CREATE X-NODEREF statement

Creates a handle which can be used as a parameter or return-value for methods which will associate the handle with an XML node. This object is not a node in its own right, but merely a way to provide access to the underlying XML node.

Syntax

```
CREATE X-NODEREF handle [ IN WIDGET-POOL widget-pool-name ]
```

handle

A valid X-noderef object handle to use for the new XML node.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates, at run time, to the name of the widget pool that contains the XML node.

Note: Widget pool names are not case-sensitive.

An X-noderef object is not in a fully usable state until it has been associated with an X-document. Before this, you can only access its HANDLE, TYPE, PRIVATE-DATE, and UNIQUE-ID attributes without error. To avoid run-time errors, the code below shows one way to test the status of an X-noderef object (*X-noderef*):

```
cType = X-noderef:SUBTYPE NO-ERROR.
IF VALID-HANDLE(X-noderef) AND cType = 'ELEMENT' THEN DO:
   /*** your code ***/
END.
```

Example

The following code fragment depicts creating an XML document node reference and using it to create a node:

```
DEFINE VARIABLE hXdoc AS HANDLE NO-UNDO.
. . .
CREATE X-DOCUMENT hXdoc.
CREATE X-NODEREF hXnode.
hXdoc:CREATE-NODE(hXnode,"City","ELEMENT").
. . .
```

See also

CREATE X-DOCUMENT statement, DELETE OBJECT statement, X-document object handle, X-noderef object handle

CURRENT-CHANGED function

Returns TRUE if the copy of the record in the buffer after executing a FIND CURRENT or GET CURRENT differs from the copy of the record in the buffer before executing the FIND CURRENT or GET CURRENT. That is, if the current application changes the record, but no other user changes the record during its scope in the current application, CURRENT-CHANGED returns FALSE.

Syntax

```
CURRENT-CHANGED record
```

record

The name of a table or buffer.

Example

The following example finds the first customer record with NO-LOCK and makes it available to the user to review and change:

r-currch.p

```
FORM Customer.Name Bustomer.Balance WITH FRAME upd.

ON GO OF FRAME upd DO:
  DO TRANSACTION:
    FIND CURRENT Customer EXCLUSIVE-LOCK.
    IF CURRENT-CHANGED Customer THEN DO:
      MESSAGE "This record has been changed by another user" SKIP
        "Please re-enter your changes." VIEW-AS ALERT-BOX.
      DISPLAY Customer.Name Customer.Balance WITH FRAME upd.
      RETURN NO-APPLY.
    END.
    ASSIGN Customer.Name Customer.Balance.
  END.
  FIND CURRENT Customer NO-LOCK.
END.

FIND FIRST Customer NO-LOCK.
DISPLAY Customer.Name Customer.Balance WITH FRAME upd.
DO ON ENDKEY UNDO, LEAVE:
  ENABLE Customer.Name Customer.Balance WITH FRAME upd.
  WAIT-FOR "GO" OF FRAME upd.
END.
```

While the user reviews the record, other users can change it. After the user makes a change of their own and enters GO in the frame, the first FIND CURRENT statement refetches the current customer record with an EXCLUSIVE-LOCK (preventing other users from reading or updating it). Then, the CURRENT-CHANGED function compares the contents of the customer record with the copy of the customer record that was in the buffer before the FIND CURRENT statement. If it differs, the CURRENT-CHANGED function returns a TRUE value, prints a message, and displays the contents of the customer record contained in the buffer. The RETURN NO-APPLY option prevents the program from ending and gives the user another chance to change the customer record.

The CURRENT-CHANGED function returns a FALSE value if the copy of the customer record that is in the buffer was not modified. After verifying that the copy of the record has not changed, the ASSIGN statement updates the customer record and a second FIND CURRENT statement down grades the record to NO-LOCK. Thus, while the user has ample time to review and change the record, the actual transaction time is kept to a minimum to allow other users access.

Notes

- The CURRENT-CHANGED function is valid only when called after a FIND CURRENT or GET CURRENT statement.
- If a client application modifies the buffer, the AVM compares the newly read record with the buffer contents from that application, rather than with the record read from the server. The CURRENT-CHANGED function continues to return a value based on the contents of the buffer until the next FIND CURRENT or GET CURRENT operates on that buffer or until the buffer goes out of scope or is released.
- The CURRENT-CHANGED function can compare the current values with the initial values of BLOB or CLOB fields provided the table is in an OpenEdge database.

See also

FIND statement, GET statement, LOCKED function

CURRENT-LANGUAGE function

Returns the current value of the CURRENT-LANGUAGE variable.

Note: Does not apply to SpeedScript programming.

Syntax

```
CURRENT-LANGUAGE
```

Example

The following example displays a message indicating the setting of your CURRENT-LANGUAGE:

r-curlng.p

```
DEFINE VARIABLE cur-lang AS CHARACTER NO-UNDO.

cur-lang = CURRENT-LANGUAGE.

IF cur-lang = "?" THEN
  MESSAGE "Your current language is not set.".
ELSE
  MESSAGE "Your current language is" cur-lang.
```

Notes

- An r-code file may contain several text segments each associated with a different language. The setting of the CURRENT-LANGUAGE variable determines from which r-code text segment the AVM reads character-string constants.
- If the value of CURRENT-LANGUAGE is a quoted question mark ("?"), the AVM reads character-strings from the default text segment.
- The value of CURRENT-LANGUAGE might be a comma-separated list of language names. If so, the AVM searches r-code for a text segment that matches the first language in the list. If that segment is not found, then it searches for a text segment for the next entry in the list until a segment is found.
- You can initialize the CURRENT-LANGUAGE variable with the Language (-lng) parameter.
- The behavior of CURRENT-LANGUAGE when one procedure calls another is as follows:
  - If a procedure changes the value of CURRENT-LANGUAGE, calls from the procedure to the CURRENT-LANGUAGE function return the name of the new language, but the procedure continues to use the character strings of the original language.
  - If the procedure then runs another procedure, when the called procedure gets control, calls from the called procedure to the CURRENT-LANGUAGE function return the name of the new language, and the called procedure uses the character strings of the new language.
  - When the called procedure finishes and control returns to the original procedure, calls from the original procedure to the CURRENT-LANGUAGE function return the name of the new language, but the original procedure continues to use the character strings of the original language.

See also

COMPILE statement, CURRENT-LANGUAGE statement

CURRENT-LANGUAGE statement

Sets the CURRENT-LANGUAGE variable for the current ABL session.

Note: Does not apply to SpeedScript programming.

Syntax

```
CURRENT-LANGUAGE = string-expression
```

string-expression

A character-string expression that specifies a language name or a comma-separated list of language names.

Example

This example procedure uses the CURRENT-LANGUAGE function to find the current language, prompts the user to choose a new language, and then uses the CURRENT-LANGUAGE statement to reset and display the name of the new current language:

r-chglng.p

```
DEFINE VARIABLE cur-lang AS CHARACTER FORMAT "x(10)" VIEW-AS RADIO-SET
    RADIO-BUTTONS czech,   "Czech",
                  danish,  "Danish",
                  dutch,   "Dutch",
                  english, "English",
                  french,  "French",
                  german,  "German",
                  hungar,  "Hungarian",
                  italian, "Italian",
                  norweg,  "Norwegian",
                  polish,  "Polish",
                  portug,  "Portuguese",
                  swedish, "Swedish".

cur-lang = IF CURRENT-LANGUAGE = "?" THEN "English" ELSE CURRENT-LANGUAGE.

UPDATE cur-lang NO-LABELS.

CURRENT-LANGUAGE = cur-lang.
MESSAGE "New language is" CURRENT-LANGUAGE.
```

Notes

- The value of CURRENT-LANGUAGE might be a comma-separated list of language names. If so, the AVM searches r-code for a text segment that matches the first language in the list. If that segment is not found, then it searches for a text segment for the next entry in the list until a segment is found.

- You can initialize the CURRENT-LANGUAGE variable with the Language (-lng) parameter.
- The behavior of CURRENT-LANGUAGE when one procedure calls another is as follows:
  - If a procedure changes the value of CURRENT-LANGUAGE, calls from the procedure to the CURRENT-LANGUAGE function return the name of the new language, but the procedure continues to use the character strings of the original language.
  - If the procedure then runs another procedure, when the called procedure gets control, calls from the called procedure to the CURRENT-LANGUAGE function return the name of the new language, and the called procedure uses the character strings of the new language.
  - When the called procedure finishes and control returns to the original procedure, calls from the original procedure to the CURRENT-LANGUAGE function return the name of the new language, but the original procedure continues to use the character strings of the original language.

See also

COMPILE statement, CURRENT-LANGUAGE statement

CURRENT-RESULT-ROW function

Returns the number of the current row of a specified query as an INTEGER value.

Syntax

```
CURRENT-RESULT-ROW ( query-name )
```

query-name

A character expression that evaluates to the name of a currently open, scrolling query. If *query-name* does not resolve to the name of a query, or if the query is not open or not scrolling, then the function returns the Unknown value (?).

Note: Searching for a query using a handle is more efficient than a character expression. The AVM resolves a character expression at run time by searching in the current routine for a static query with that name. If not found, the AVM searches the enclosing main procedure. If still not found, the AVM searches up through the calling programs of the current routine and their main procedures. Since a handle uniquely identifies a query, no such search is required.

Example

The following example uses the QUERY-OFF-END function to determine when to leave the REPEAT loop:

r-resrow.p

```
DEFINE QUERY cust-query FOR Customer SCROLLING.
OPEN QUERY cust-query FOR EACH Customer WHERE Customer.Country = "USA".

REPEAT:
  GET NEXT cust-query.
  IF QUERY-OFF-END("cust-query") THEN LEAVE.
  DISPLAY CURRENT-RESULT-ROW("cust-query") LABEL "Result Row"
    Customer.CustNum Customer.Name.
END.
```

Notes

- To use the CURRENT-RESULT-ROW function with a query, the query must be associated with a browse widget or you must define the query with the SCROLLING option. For more information on query definitions, see the reference entry for the DEFINE QUERY statement.
- If the query is empty, CURRENT-RESULT-ROW returns the Unknown value (?).
- If the query is positioned before the first record, CURRENT-RESULT-ROW returns the value 1. If the query is positioned beyond the last record, CURRENT-RESULT-ROW returns a value 1 greater than the number of rows in the query result list.
- When possible, the AVM performs optimizations for GET LAST and REPOSITION statements. These optimizations make the results list invalid. At that point, CURRENT-RESULT-ROW returns the Unknown value (?). These optimizations do not occur if the query is opened with the PRESELECT option or has an associated browse widget.

See also

CLOSE QUERY statement, CURRENT-REQUEST-INFO attribute, DEFINE BROWSE statement, DEFINE QUERY statement, GET statement, NUM-RESULTS function, OPEN QUERY statement, REPOSITION statement

CURRENT-VALUE function

Returns the current INT64 value of a sequence defined in the Data Dictionary.

Syntax

```
CURRENT-VALUE ( sequence [ , logical-dbname ] [ , tenant-id ] )
```

sequence

An identifier that specifies the name of a sequence defined in the Data Dictionary.

logical-dbname

An identifier that specifies the logical name of the database in which the sequence is defined. The database must be connected. You can omit this parameter if the sequence name is unambiguous. If a sequence with this name exists in more than one connected database, then you must specify *logical-dbname*.

tenant-id

An integer expression that evaluates to the tenant ID of a regular tenant, including the default tenant (0). This option applies only to a multi-tenant sequence specified by *sequence* and is intended for access primarily by a super-tenant user.

If you are a regular-tenant user and you specify *tenant-id*, the specified tenant must be the same as the tenant associated with the database connection identity for *logical-dbname* or the AVM raises a run-time error.

If you are a super-tenant user and you do not specify *tenant-id*, the function assumes the effective tenant ID (GET-EFFECTIVE-TENANT-ID function).

If the sequence specified by *sequence* is non-multi-tenant (a shared sequence in a multi-tenant database), and you specify *tenant-id*, the option is ignored.

Example

The following example finds the current value of the NextCustNum sequence and then looks for Orders with that Customer number:

r-curval.p

```
DEFINE VARIABLE cur-cust NO-UNDO LIKE Customer.CustNum.

cur-cust = CURRENT-VALUE(NextCustNum).
IF CAN-FIND(FIRST Order WHERE Order.CustNum = cur-cust) THEN
FOR EACH Order NO-LOCK WHERE Order.CustNum = cur-cust,
  EACH OrderLine OF Order NO-LOCK
  BREAK BY Order.OrderNum:

  IF FIRST-OF(Order.OrderNum) THEN
    DISPLAY Order.OrderNum Order.OrderDate Order.CustNum
      WITH FRAME order-info CENTERED ROW 2 1 COL.

  DISPLAY OrderLine.
END.
ELSE DO:
  FIND FIRST Customer WHERE Customer.CustNum = cur-cust NO-LOCK NO-ERROR.
  IF AVAILABLE Customer THEN
    MESSAGE "No Orders Exist for Customer " + Customer.Name +
      ", " + STRING(Customer.CustNum)
      VIEW-AS ALERT-BOX INFORMATION BUTTONS OK TITLE "No Orders".
  ELSE
  MESSAGE "Customer number" cur-cust "does not exist."
    VIEW-AS ALERT-BOX INFORMATION BUTTONS OK TITLE "No Customer".
END.
```

Notes

- The current value of a sequence can be one of the following:
  - The initial value specified in the Data Dictionary
  - The last value set with either the CURRENT-VALUE statement or the NEXT-VALUE function
  - The Unknown value (?) if the sequence has exceeded its minimum or maximum and is not cycling
- If *sequence* is a multi-tenant sequence in the database, each regular tenant has their own current value of the sequence. So, the same values are returned for each tenant that invokes this function. If the sequence is shared in a multi-tenant database, the values returned by this function are unique across all tenants in the database.

Caution: Be very careful using multi-tenant sequences when you are updating shared tables or multi-tenant tables for a tenant group. For more information, see the sections on using multi-tenant-enabled sequences in *OpenEdge Programming Interfaces*.

- Sequence values are stored in the database in which they are defined, and persist between each invocation of the CURRENT-VALUE statement or NEXT-VALUE function.
- You cannot invoke the CURRENT-VALUE function from within a WHERE clause. Doing so generates a compiler error. To use a result from the CURRENT-VALUE function in a WHERE clause, assign the result to a variable, then use the variable in the WHERE clause.
- You can use any combination of the NEXT-VALUE function, CURRENT-VALUE function, CURRENT-VALUE statement, and their dynamic versions. Use the dynamic version when you don't know what the database name or sequence name is at compile time.
- Be careful when accessing a database sequence with an alias that points to a different database than the one used when the alias was defined. If you supply an alias name to the CURRENT-VALUE function or the NEXT-VALUE function, only the database used to define the alias is referenced. In this case, it is preferable to use the DYNAMIC-CURRENT-VALUE function and DYNAMIC-NEXT-VALUE function instead of the CURRENT-VALUE function and NEXT-VALUE function, respectively.

See also

CURRENT-VALUE statement, DYNAMIC-CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, DYNAMIC-NEXT-VALUE function, NEXT-VALUE function

CURRENT-VALUE statement

Resets the current integer value of a sequence defined in the Data Dictionary.

Syntax

```
CURRENT-VALUE ( sequence [ , logical-dbname ] [ , tenant-id ] ) = expression
```

sequence

An identifier that specifies the name of a sequence defined in the Data Dictionary.

logical-dbname

An identifier that specifies the logical name of the database in which the sequence is defined. The database must be connected. You can omit this parameter if the sequence name is unambiguous. If more than one connected database has a sequence with given name, then you must supply *logical-dbname*.

tenant-id

An integer expression that evaluates to the tenant ID of a regular tenant, including the default tenant (0). This option applies only to a multi-tenant sequence specified by *sequence* and is intended for access primarily by a super-tenant user.

If you are a regular-tenant user and you specify *tenant-id*, the specified tenant must be the same as the tenant associated with the database connection identity for *logical-dbname* or the AVM raises a run-time error.

If you are a super-tenant user and you do not specify *tenant-id*, the function assumes the effective tenant ID (GET-EFFECTIVE-TENANT-ID function).

If the sequence specified by *sequence* is non-multi-tenant (a shared sequence in a multi-tenant database), and you specify *tenant-id*, the option is ignored.

expression

An integer expression assigned as the current value of the specified sequence. If *expression* is outside the boundary set by the initial value (at one end) and the lower limit or upper limit (at the other end) for the sequence, the AVM returns an error, and the sequence value remains unchanged.

Note: Compound assignment operators (+=, -=, *=, /=) can be used with the CURRENT-VALUE statement. For more information, see += Addition assignment operator, -= Subtraction assignment operator, *= Multiplication assignment operator, and /= Division assignment operator.

Example

The following example resets the current value of the NextCustNum sequence to the CustNum value of the last Customer record if that is a valid value for the sequence:

r-curvl1.p

```
FIND LAST customer NO-LOCK.

IF Customer.CustNum < CURRENT-VALUE(NextCustNum) AND
  Customer.CustNum > 1000 THEN DO:
  CURRENT-VALUE(NextCustNum) = Customer.CustNum.
  MESSAGE "The value of NextCustNum has been changed to"
    Customer.CustNum VIEW-AS ALERT-BOX INFORMATION BUTTONS OK.
END.
ELSE
  MESSAGE "The value of NextCustNum remains"
    CURRENT-VALUE(NextCustNum) VIEW-AS ALERT-BOX INFORMATION BUTTONS OK.
```

Notes

- The user must have CAN-WRITE privileges on the _Sequence table to use the CURRENT-VALUE statement.
- The value of a sequence set by the CURRENT-VALUE statement persists in the database until the next CURRENT-VALUE statement or NEXT-VALUE function is invoked for the sequence, or until the sequence is deleted from the database.
- You cannot set a sequence to the Unknown value (?).
- If *sequence* is a multi-tenant sequence in the database, each regular tenant has their own current value of the sequence. So, each tenant that invokes this statement can set their own current values, which can be the same or different than the current values set by other tenants. If the sequence is shared in a multi-tenant database, any current value set by this statement is shared by all tenants in the database.

  Caution: Be very careful using multi-tenant sequences when you are updating shared tables or multi-tenant tables for a tenant group. For more information, see the sections on using multi-tenant-enabled sequences in *OpenEdge Programming Interfaces*

- You can use any combination of the NEXT-VALUE function, CURRENT-VALUE function, CURRENT-VALUE statement, and their dynamic versions. Use the dynamic version when you don't know what the database name or sequence name is at compile time.

See also

CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, DYNAMIC-NEXT-VALUE function, NEXT-VALUE function

Data types

The data type of a data element defines what kind of data the data element can store. ABL supports the following basic kinds of data types:

- ABL built-in primitive types, including mappings to corresponding .NET primitive types
- Object types, which include both ABL and supported .NET object types, including both built-in and user-defined class, enumeration, and interface types
- ABL handle-based objects
- ABL arrays, including one-dimensional arrays of ABL primitive types, ABL object types, or .NET object types

ABL primitive types (see Table 21: ABL primitive types) are built-in data types that can hold values with relatively simple content and that support relatively simple operations that can typically be applied using built-in ABL operands, such as arithmetic or relational operands (for example, the + Addition operator, or the EQ or = operator). The values for all ABL primitive types, other than the MEMPTR, BLOB, CLOB, and LONGCHAR, are limited in memory size to 32KB. BLOB and CLOB fields can be up to 1GB in size. MEMPTR and LONGCHAR variables can be up to 2GB - 1 byte in size, but are limited by available memory.

Note that you can define BLOB and CLOB data types only for table or temp-table fields, and, unlike most other ABL primitive types, the operations they support do not have built-in ABL operands, but are available using built-in ABL functions (for example, the COPY-LOB statement). Note that `COPY-LOB` file operations involving LONGCHAR variables are limited to 1GB - 1 byte.

You can use a primitive data type keyword in the following ABL syntax:

- DEFINE PARAMETER statement
- DEFINE PROPERTY statement
- DEFINE VARIABLE statement
- Fields of a table from an OpenEdge RDBMS using Progress Developer Studio for OpenEdge or the Data Dictionary.
- Fields of a temp-table using the DEFINE TEMP-TABLE statement or methods of a Temp-table object handle.
- FUNCTION statement (return type)
- METHOD statement (return type)
- Parameter definition syntax (for a user-defined method or function)

ABL object types (see Table 22: ABL complex types) are complex types that function according to object-oriented principles. They can be broadly divided into two categories:

- **Classes** — ABL supports a set of built-in object types and also allows you to create your own user-defined object types using using the CLASS statement and the INTERFACE statement. All ABL class types ultimately inherit from the ABL root class, Progress.Lang.Object class. Each object type encapsulates a set of data

and behavioral elements (members). ABL class members can include implementations for data members, properties, and methods.

You can create instances of classes (*class-based objects*) at run time using the NEW function (classes), and you can reference each instance and its PUBLIC members using an *object reference*, which is an ABL data element defined to reference a specific object type. You can define an object reference for any kind of ABL data element that you can define as an ABL primitive type (except a database table field). However, you can define a field of a temp-table as an object reference to the ABL root class (Progress.Lang.Object class).

- **Enumerations** — An enumeration (also known as enumerated types or enums) is made up of a list of strongly typed, named constants called members. The value of a variable defined as an enumeration is restricted to the list of members defined for that enumeration. As with classes, ABL supports a set of built-in enumeration types and also allows you to create your own user-defined enumerations using the ENUM statement. You can define two types of enum. In the standard version, a variable defined as an enum can only be assigned one value at a time. You can also define a flag enum. Any variable defined as a flag enum can be assigned any combination of the enum's members at one time.

  Enumerations are implicitly class based, but unlike classes defined using the CLASS statement, enumeration types cannot be explicitly instantiated using the NEW function. To assign a value to a variable, property, or parameter defined as an enumeration type, you use Type-name syntax to access the members defined in the enumeration type. See the Enumeration member access reference entry for more information.

For information on the built-in ABL class, interface, and enumeration types, see the Class, Interface, and Enumeration Reference.

.NET primitive types (see Table 24: Implicit mappings between .NET and ABL data types) include data types that are built-into .NET languages. (Note that each language uses its own nomenclature. Compare, for example, the C# int versus the Visual Basic Integer.) .NET also aliases (*maps*) a standard set of object types (*.NET mapped object types*, for example, `System.Int32`) to these primitive types. All .NET languages can reference each such type as either the primitive type that the particular language supports or as the alias for the corresponding .NET object type that every language supports.

ABL also references both the .NET primitive types and their corresponding mapped object types by mapping each ABL primitive type to a given set of .NET primitive and object type mappings (see Table 24: Implicit mappings between .NET and ABL data types). Thus, ABL documentation refers collectively to both the .NET primitive types and the corresponding .NET mapped object types as *.NET mapped data types*.

.NET object types (see Table 22: ABL complex types) include all class types (and their derivatives) that derive from the .NET root class, `System.Object`, and .NET interface types, which can inherit from other .NET interface types, but otherwise function for a class much like an ABL interface type. You can reference a .NET object type like an ABL object type, by using an ABL object reference defined as that object type, or by referencing members of a .NET class. .NET object types also consist of two basic kinds of types:

- **Value types** — Objects that .NET creates, passes, and assigns by value. Value type objects all inherit from the .NET class, `System.ValueType`. In ABL, when you access a value type from .NET, you access a new copy of the object that is separate from the one that is maintained by .NET. If you then change object data in ABL, these changes do not appear in any copy of the object maintained by .NET. In addition, the ABL object reference to the ABL copy of the object has no affect on the .NET garbage collection of any .NET copy of the object.
- **Reference types** — Objects that .NET creates, passes, and assigns by reference. In ABL, when you access a .NET reference type, you access the same copy of the object that is maintained by .NET. If you then change object data in ABL, these changes also appear in .NET, because .NET references the same object. In addition, the ABL object reference is counted as a reference to the object for .NET garbage collection.

ABL also provides limited support for .NET abstract classes and .NET generic types. A .NET abstract class is similar to an ABL abstract class. A generic type has a type definition that can function as one of several different types, depending on type parameters used to complete the effective type name. .NET generic types are briefly described further in this entry. For more information on the basic kinds of .NET object types that ABL supports, see the notes section of this reference entry. OpenEdge also provides a set of built-in .NET class and interface types to support access to .NET object types. For information on the built-in .NET class and interface types, see the Class, Interface, and Enumeration Reference.

Within certain restrictions, an ABL class can inherit from a .NET class and implement .NET interfaces, similar to inheriting from an ABL class or implementing an ABL interface, respectively. When an ABL class inherits from a .NET class, any of its methods that override methods in the .NET class hierarchy can be called polymorphically on the .NET super class from **both** ABL and .NET. As a result, any ABL class that inherits from a .NET class becomes an *ABL-derived .NET class*. In fact, when an ABL-derived .NET class is instantiated in an ABL session, an instance with a corresponding .NET class type is also instantiated in the .NET context with reference to any ABL-overridden methods.

Similarly, when an ABL class implements a .NET interface, all of the ABL-implemented properties and methods can be called polymorphically on the interface type from **both** ABL and .NET. As a result, any ABL class that implements a .NET interface becomes an *ABL-extended .NET class*. In fact, when an ABL-extended .NET class that implements .NET interfaces is instantiated in an ABL session, an instance with a corresponding .NET class type is also instantiated in the .NET context with reference to the ABL-implemented properties and methods that might be accessed from .NET on each of the implemented interface types.

Note: An ABL-derived .NET class is also considered an ABL-extended .NET class. However, an ABL-extended .NET class that only implements .NET interfaces is not an ABL-derived .NET class. Thus, when OpenEdge documentation refers to an ABL-extended .NET class, it can also (but not necessarily) be referring to an ABL-derived .NET class.

ABL handle-based objects (see Table 22: ABL complex types) include a set of complex, weakly-typed objects, some of which ABL provides as built-in system objects (such as, the SESSION system handle), and others that ABL supports as a pre-defined set of objects that you can create as needed (such as, a FRAME widget or a record buffer). Handle-based objects exist independently and have no inheritance hierarchy like class-based objects. However, like class-based objects, handle-based objects have members consisting of a set of attributes (data) and methods (behavior). Depending on the object, you can create a handle-based object as a compile-time (static) object using an appropriate DEFINE statement or as a run-time (dynamic) object using an appropriate CREATE statement or other executable statement. You can reference system objects using the built-in system handle (keyword) pre-defined for them. You can reference static handle-based objects by name, using appropriate syntax for each type, and you can reference dynamic or static handle-based objects using a common primitive data element known as a handle, which you define as the HANDLE data type. Because of the weak typing of these objects, you can reference all static and dynamic handle-based objects that you define or create using the same handle. ABL also provides some system handles that provide access to particular types of pre-defined handle-based objects that are in a given state (such as, the CURRENT-WINDOW system handle for accessing a particular WINDOW widget).

ABL arrays (see Table 22: ABL complex types) are limited to one dimension and can include elements of any primitive or object type that you can define for a variable (see the DEFINE VARIABLE statement reference entry). ABL also provides support for mapping ABL arrays to one-dimensional .NET array objects of the same element type. This means that while you can access all .NET arrays as class instances, you can also make direct array assignments and pass routine parameters between ABL arrays and equivalent one-dimensional .NET array objects. This also includes .NET arrays whose elements are .NET mapped data types (referred to as *.NET arrays of mapped types*), where assignments to or from ABL arrays work using the rules of implicit data type mapping. The element types supported for a .NET array of mapped types are identical to the .NET data types that ABL implicitly maps to ABL primitive types (see Table 24: Implicit mappings between .NET and ABL data types). For more information on support for both ABL arrays and .NET arrays, see the notes section of this reference entry.

The following table describes the primitive types supported in ABL.

Table 21: ABL primitive types

Primitive type	Description
BLOB	BLOB (Binary Large OBject) specifies a database table or temp-table field that contains a BLOB locator, which points to the associated BLOB data stored in the database. You must use a MEMPTR to manipulate the binary contents of a BLOB field in ABL.
CHAR	CHAR data consists of numbers, letters, and special characters. CHAR is a synonym for the CHARACTER data type.
CHARACTER	CHARACTER data consists of numbers, letters, and special characters.
CLOB	CLOB (Character Large OBject) specifies a database table or temp-table field that contains a CLOB locator, which points to the associated CLOB data stored in the database. You must use a LONGCHAR to manipulate the character contents of a CLOB field in ABL.
COM-HANDLE	A COM-HANDLE is a handle to a COM object (ActiveX Automation object or ActiveX Control).
DATE	DATE fields contain dates.
DATETIME	DATETIME data has two parts: an ABL date and an ABL time. The unit of time is milliseconds from midnight.
DATETIME-TZ	DATETIME-TZ data has three parts: an ABL date, an ABL time, and an integer representing the time zone offset from Coordinated Universal Time (UTC). The unit of time is milliseconds from midnight. The unit of time zone offset is minutes.
DECIMAL	DECIMAL data consists of decimal numbers up to 50 digits in length including up to 10 digits to the right of the decimal point.
HANDLE	A HANDLE is a pointer to an ABL handle-based object. This can be a compile-time defined static object or a run-time defined dynamic object. **Note:** HANDLE and WIDGET-HANDLE can be assigned to each other and used interchangeably. WIDGET-HANDLE is supported only for backward compatibility.
INT	An INT consists of 32-bit data (whole numbers). INT is a synonym for the INTEGER data type.
INT64	An INT64 consists of 64-bit data (whole numbers).
INTEGER	An INTEGER consists of 32-bit data (whole numbers).
LOGICAL	LOGICAL data evaluates to TRUE or FALSE (or YES or NO).
LONGCHAR	A LONGCHAR consists of CHARACTER data that can be up to 2GB - 1 byte in size. You can use a LONGCHAR to manipulate the character contents of a CLOB database or temp-table field in ABL.

Primitive type	Description
MEMPTR	A MEMPTR contains a sequence of bytes in memory. You can use a MEMPTR to manipulate the contents of a BLOB database or temp-table field in ABL.
RAW	RAW data can be any kind of data, even data from non-OpenEdge databases. It is not converted in any way.
RECID	A RECID is a unique internal identifier for a record within a single database storage area. **Note:** RECID is supported mainly for backward compatibility. For most applications, use ROWID instead.
ROWID	A ROWID is a unique internal identifier for a record within a single database storage area. For partitioned tables, it is also unique across all partitions of a given table.
WIDGET-HANDLE	A WIDGET-HANDLE is a pointer to an ABL handle-based object. **Note:** HANDLE and WIDGET-HANDLE can be assigned to each other and used interchangeably. WIDGET-HANDLE is supported only for backward compatibility.

The following table describes the non-primitive (*complex*) types supported in ABL.

Table 22: ABL complex types

Complex type	Description
Array	An ABL array type is complex type that specifies a one-dimensional array of elements of the same scalar data type with a 1-based index, where a *scalar* data type is any data type that is not, itself, an array. The elements of an ABL array can contain scalars of any supported ABL primitive type, any ABL object type, or any .NET object type (other than a .NET mapped object type). The type definition for an ABL array is specified by the type definition for the array element with the addition of the EXTENT option (or with the Extent option selected in OpenEdge database tools). Thus, you can define an ABL array data element using similar features and syntax used to define primitive and object-type data elements. However, you cannot define ABL array types for the BLOB or CLOB primitive type. Also, note that unlike ABL arrays, .NET arrays are objects with an object type, just like any other .NET type. However, all .NET array object types derive from the `System.Array` class (a reference type), and you can create and access .NET arrays using public members of `System.Array`. Also, .NET arrays can be multi-dimensional and typically have a 0-based index. In .NET, the object type name of an array object is the object type name of its array elements appended with a set of square brackets (`[]`) with an embedded comma added for each additional dimension in the array. For example, `System.Drawing.Point[]` is a one-dimensional array object type and `System.Drawing.Point[,]` is a two-dimensional array object type. .NET languages support syntax for additional kinds of array objects, including jagged arrays. For more information on .NET array syntax, see the "Arrays Tutorial" in the *C# Programmer's Reference* on MSDN. In ABL, you must also enclose any .NET array object type name in double-quotes in order to handle the square brackets and any commas, which are special characters in ABL names, for example, `"System.Drawing.Point[]"`. For more information on specifying array types, see the Type-name syntax reference entry.
Class	Classes encapsulate a set of data and behavioral elements (members), including implementations for data members, properties, and methods. Class definitions can also implement interfaces, which define a common set of prototypes for methods, properties, and events that classes can implement. For more information on classes and interfaces, see *Develop Object-oriented ABL Applications*.

Complex type	Description
Enumeration	An enumeration is made up of a list of strongly typed, named constants called members. The value of a variable defined as an enumeration type is restricted to the list of members defined for that enumeration type.
Handle-based object	A handle-based object has a built-in and inherent ABL type that provides data and behavior of varying complexity depending on the purpose of the object. A few examples include: • Visual representation objects (widgets), such as buttons (defined using the DEFINE BUTTON statement, rectangles (defined using the DEFINE RECTANGLE statement statement), or data-representation widgets, such as a fill-ins (defined as part of the DEFINE VARIABLE statement or DEFINE PARAMETER statement) or browses (grid-like widgets defined using the DEFINE BROWSE statement) • Data objects, such as temp-tables (defined using the DEFINE TEMP-TABLE statement), ProDataSets, (defined using the DEFINE DATASET statement), and related data objects, such as queries (defined using the DEFINE QUERY statement) • Streams (defined using the DEFINE STREAM statement), which are used for reading and writing sequential data, such as text files • Procedure objects (each defined as a file of ABL source code and created at run time using the RUN statement) • Socket and server socket objects (created at run time using the CREATE SOCKET statement and CREATE SERVER-SOCKET statement **Note:** While each type of handle-based object is unique, because of their weak typing, you can reference all such objects using the same primitive type, HANDLE (see Table 21: ABL primitive types). For more information on each type of handle-based object, see the reference entry for its type in the Widget Reference or the Handle Reference, and see the reference entry for its respective DEFINE, CREATE, or other instantiating executable statement.

The following table lists the default data formats and initial values for ABL primitive and object types.

Table 23: Default ABL data type initial values and display formats

Data type	Default initial value	Default display format
BLOB[15, 17]	Unknown value (?)	See footnote [15].
CHARACTER	"" (an empty string)	X(8)

[15] You cannot display a BLOB, CLOB, MEMPTR, RAW, or ROWID value directly. However, you can convert a MEMPTR, RAW, or ROWID value to a character string representation using the STRING function and display the result. You can also convert a BLOB to a MEMPTR, and then use the STRING function. A MEMPTR or RAW value converts to a decimal integer string. A ROWID value converts to a hexadecimal string, "0x*hexdigits*", where *hexdigits* is any number of characters "0" through "9" and "A" through "F". You can display a CLOB field by converting it to a LONGCHAR, and displaying the LONGCHAR using the VIEW-AS EDITOR LARGE phrase only.

Data type	Default initial value	Default display format
CLASS[16, 17],	Unknown value (?)	See footnote [15] .
CLOB[15, 17]	Unknown value (?)	See footnote [15].
COM-HANDLE[17]	Unknown value (?)	>>>>>>9
DATE	Unknown value (?) (displays as blanks)	99/99/99
DATETIME	Unknown value (?)	99/99/9999 HH:MM:SS.SSS
DATETIME-TZ	Unknown value (?)	99/99/9999 HH:MM:SS.SSS+HH:MM
DECIMAL	0	->>,>>9.99
HANDLE[17]	Unknown value (?)	>>>>>>9
INT64	0	->,>>>,>>9
INTEGER	0	->,>>>,>>9
LOGICAL	no	yes/no
LONGCHAR[15]	"" (an empty string)	See footnote [15].
MEMPTR[15, 17]	A zero-length sequence of bytes	See footnote [15].
RAW[15, 17]	A zero-length sequence of bytes	See footnote [15].
RECID	Unknown value (?)	>>>>>>9
ROWID[15, 17]	Unknown value (?)	See footnote [15].

For more information on using the built-in ABL primitive types, see *Develop ABL Applications* and the Web paper, *ABL Data Types*.

As noted previously in this entry, ABL supports references to .NET types in two basic ways:

1. You can make direct and explicit reference to .NET object types using similar syntax that is supported for referencing ABL user-defined class and interface types. For supported .NET object types, this includes both the instantiation of a .NET class in ABL and the derivation of the .NET class by an ABL user-defined class (ABL-derived .NET class), and it includes the implementation of supported .NET interfaces by an ABL user-defined class (ABL-extended .NET class). To integrate the .NET class hierarchy with the ABL class hierarchy, ABL views System.Object as an immediate subclass of the ABL root class (Progress.Lang.Object

[16] If you display a class instance using the MESSAGE statement, ABL automatically invokes the ToString( ) method (Object) (provided by the Progress.Lang.Object class) on the object reference. To display a class instance in a frame (for example, using the DISPLAY statement), you must first explicitly convert the object reference to a displayable type using the INT64 function, the INTEGER function, the STRING function, or the ToString( ) method (Object).

[17] You cannot use the INITIAL option to specify an initial value for this data type as part of the definition of a variable, procedure parameter, or class-based property.

class). In this way, you can manage .NET object types in ABL using many of the same mechanisms that you use for managing ABL class and interface types. However, you must observe the following limitations:

- You cannot directly reference any .NET object type that is supported as a .NET mapped data type, except to define a .NET array of such types. You can only reference .NET mapped data types as their equivalent ABL built-in primitive types. For more information on .NET mapped data types, see the immediately following Step 2.
- You cannot use `System.Threading.Thread`, or any derived class—ABL is single-threaded.
- You cannot use `System.MulticastDelegate`, or any derived class (otherwise referred to as delegates) to provide handlers for .NET events. ABL provides its own event handling model for .NET events. For more information, see the Class Events Reference.
- You cannot define an ABL interface that inherits from a .NET interface.
- ABL imposes additional requirements on the .NET classes you can extend and the .NET interfaces you can implement in an ABL user-defined class. For more information, see the CLASS statement and INTERFACE statement reference entries.

For more information on the requirements for accessing .NET object types, see the notes section of this reference entry and *Use GUI for .NET Objects in ABL Applications*.

2. You can make implicit access to all .NET primitive types and their associated mapped object types by using the ABL built-in primitive types that are mapped to them. Because .NET mapped object types and .NET primitive types, together, represent the complete set of .NET mapped data types, the implicit mapping between .NET mapped data types and ABL primitive types allows you to access .NET method parameters, fields (data members), and properties using the corresponding ABL primitive types without direct reference to their .NET data type equivalents. In fact, ABL does not allow you to directly reference either the .NET primitive types or the .NET mapped object types as scalars without raising a compile-time error. (The exception is when defining a .NET array of mapped types. For more information, see the notes section of this reference entry.)

The following table shows the implicit mappings supported between .NET mapped data types and ABL built-in primitive types, showing the corresponding primitive types from C#.

Table 24: Implicit mappings between .NET and ABL data types

Implicit .NET object type	Implicit C# primitive type	ABL primitive type
`System.Boolean`	`bool`	LOGICAL
`System.Byte`	`byte`	INTEGER[18, 19]
`System.SByte`	`sbyte`	INTEGER[18]
`System.DateTime`	N/A	DATETIME

[18] An ABL INTEGER is a 32-bit number. Thus, it can hold values that are too big to store in a .NET System.Byte, System.SByte, System.Int16, or System.UInt16. Therefore, AVM raises a run-time error if an incompatible value is assigned.

[19] If you pass a negative ABL data type to an unsigned data type, the ABL virtual machine (AVM) raises a run-time error.

Implicit .NET object type	Implicit C# primitive type	ABL primitive type
`System.Decimal`	`decimal`	DECIMAL[20, 21],
`System.Int16`	`short`	INTEGER[18]
`System.UInt16`	`ushort`	INTEGER[18, 19]
`System.Int32`	`int`	INTEGER[21]
`System.UInt32`	`uint`	INT64[22, 19]
`System.Int64`	`long`	INT64[21]
`System.UInt64`	`ulong`	DECIMAL[19, 23]
`System.Double`	`double`	DECIMAL[24]
`System.Single`	`float`	DECIMAL[24]
`System.Char`	`char`	CHARACTER[25]
`System.String`	`string`	CHARACTER[21] or LONGCHAR[21, 26]

Thus, instead of using an object reference to the corresponding .NET mapped object type, you must provide or access all .NET primitive (or mapped object type) values for .NET methods, data members, and properties as ABL primitive types. Similarly, when you reference any data element or value defined as a .NET mapped data type, ABL evaluates the .NET value to its corresponding ABL primitive value. ABL checks for .NET/ABL type compatibility at compile time, except in rare cases where data type narrowing is allowed, in which case the AVM checks for data overflow or underflow at run time.

Note: To access all other .NET value types in ABL **except** mapped data types (for example, `System.Drawing.Size`), you can and must use object references to the value type objects.

[20] The range of values for a .NET System.Decimal and the range of values for an ABL DECIMAL are not equivalent. In particular, an ABL DECIMAL can be a much larger positive number or a much smaller negative number than a .NET System.Decimal can represent, and a .NET System.Decimal can represent a positive or negative number with much higher precision (with more significant digits to the right of the decimal point) than an ABL DECIMAL can represent. Therefore, the AVM raises a run-time error if you assign too large or too small of an ABL DECIMAL value to a .NET System.Decimal. If you assign too precise a .NET System.Decimal to an ABL DECIMAL, with too many significant digits to the right of the decimal point, ABL truncates the least significant digits necessary to represent the value as an ABL DECIMAL.

[21] The .NET default match for this ABL primitive type when passed as an overloaded method parameter, when passed to a System.Object parameter, when used to define an overridden .NET method, when used to implement (or override) a .NET interface (or abstract) method, property, or event, or when converted using the BOX function, all without a specified AS data type indication. For more information on AS data types, see the following paragraphs and Table 25: Explicit mappings between ABL and .NET data types.

[22] An ABL INT64 is a 64-bit number. Thus, it can hold values that are too big to store in a .NET System.UInt32. Therefore, AVM raises a run-time error if an incompatible value is assigned.

[23] An ABL DECIMAL can represent a much larger number than a System.UInt64. Therefore, AVM raises a run-time error if an incompatible value is assigned.

[24] An ABL DECIMAL represents numbers up to 50 digits long. As a result, an ABL DECIMAL value cannot represent the full range of values for a .NET System.Double or System.Single. Therefore, AVM raises a run-time error if an incompatible value is assigned. Also, an ABL DECIMAL can lose precision when it is represented by a .NET System.Double or System.Single.

[25] This ABL CHARACTER mapping supports a single Unicode character.

[26] When the value comes from .NET, ABL converts from System.String to either CHARACTER or LONGCHAR, depending on the length of the character string. When the value comes from ABL, System.String accepts values from either CHARACTER or LONGCHAR.

.NET supports a concept known as boxing. *Boxing* is the process of converting a value type (such as a C# int or .NET System.Int32) to a reference type object. Boxing a value type wraps its value inside a `System.Object`. *Unboxing* extracts the value from the `System.Object` as the original value type. In .NET, boxing and unboxing between a value type and a `System.Object` occurs during assignment or parameter passing.

So, in addition to implicitly mapping its native primitive types to their corresponding .NET mapped data types, ABL also supports boxing between its primitive or array types and a .NET `System.Object` or array object. ABL performs boxing operations automatically in two cases:

- When you assign values between a .NET `System.Object` or one-dimensional array object and a compatible ABL primitive or array type
- When you pass parameter values for .NET methods and constructors between a .NET `System.Object` or one-dimensional array object and a compatible ABL primitive or array type

However, as described further in this entry, ABL does **not** support automatic boxing operations when passing parameters to ABL routines.

When ABL does automatic boxing that involves ABL primitive types or arrays of elements containing primitive types, it also does implicit conversion between these types and the corresponding .NET mapped types (see Table 24: Implicit mappings between .NET and ABL data types). For example, if you assign an ABL INTEGER to a System.Object, ABL converts the ABL INTEGER to a `System.Int32`, which the `System.Object` accepts as a subclass value. Similarly, if you assign an ABL INTEGER array to a `System.Object`, ABL converts the ABL INTEGER array to a `"System.Int32[]"`, which the `System.Object` accepts as a subclass value. The same occurs when you pass an ABL INTEGER or INTEGER array to a `System.Object` INPUT parameter of a .NET method.

In reverse, when you assign an appropriate `System.Object` to an ABL INTEGER or INTEGER array, ABL unboxes the `System.Object` by determining the .NET mapped type that the `System.Object` represents, converts that value to its equivalent ABL primitive or primitive array value, and attempts to assign the result to the ABL INTEGER or INTEGER array (which is validated at run time). For example, if the `System.Object` represents the `System.Decimal` subclass and you are assigning it to an ABL INTEGER, ABL converts the `System.Decimal` value to an ABL DECIMAL and attempts to assign it to the ABL INTEGER.

In a similar manner, ABL also does automatic boxing directly between compatible ABL arrays and one-dimensional .NET array objects. For example, if you assign or pass .NET method parameters between a `"System.Windows.Forms.Button[]"` array object and an ABL array of `System.Windows.Forms.Button` elements, ABL automatically does the required boxing and unboxing to convert between the different array types. A similar boxing and unboxing operation occurs between an ABL primitive array and a compatible .NET array of mapped types, for example, between an ABL array of INTEGER and a .NET `"System.Int16[]"` array object. For more information on boxing and unboxing between ABL and .NET arrays, see the notes section in this reference entry on working with .NET arrays.

However in the following four ABL contexts, automatic ABL boxing or unboxing is either not supported or might not be supported as you require:

- When you use a `System.Object` directly in an expression, ABL does not unbox the `System.Object` into a compatible ABL primitive type.
- When you assign an ABL primitive value (or primitive array) to a `System.Object` and the ABL primitive type maps to multiple .NET data types, it automatically boxes the ABL primitive value (or primitive array elements) as the default matching .NET mapped object type, which might not be the .NET data type mapping that you require.
- ABL does no automatic boxing or unboxing when you pass an ABL primitive or array type to a compatible .NET object parameter of an ABL routine (ABL method, procedure, or user-defined function). Similarly, ABL also does no automatic boxing or unboxing when you pass a compatible .NET object argument to the ABL primitive or array parameter of an ABL routine.
- ABL does no boxing or unboxing of array elements when you box or unbox an ABL array. For example, ABL cannot assign between a `"System.Object[]"` and an ABL INTEGER array, because it does not

handle boxing and unboxing between the corresponding `System.Object` and the INTEGER array elements. For more information on ABL support for array assignments, see the notes section of this reference entry.

When you use a `System.Object` directly in an expression, ABL raises a compile-time error because ABL does not support automatic unboxing of a `System.Object` in an expression. Instead, you can use the ABL built-in UNBOX function in the expression to explicitly unbox the value. This function accepts the `System.Object` as input and returns an ABL primitive value that is equivalent to the .NET mapped object type value (subclass) represented by the specified .NET `System.Object` instance.

When you assign an ABL primitive value or primitive array to a `System.Object`, ABL always boxes the value or array into a particular .NET mapped type or array of mapped types, which might not be the .NET type you want. In Table 24: Implicit mappings between .NET and ABL data types, several ABL primitive types implicitly map to more than one .NET mapped data type. For each ABL primitive type that maps to multiple .NET data types, ABL uses one of these mappings as the *.NETdefault match* for the ABL primitive type (indicated by a footnote [21] in Table 24: Implicit mappings between .NET and ABL data types). Thus, when you assign an ABL primitive value or primitive array to a `System.Object`, ABL automatically boxes the value or array using the .NET default match for the specified ABL data type. For example, by default an INTEGER automatically boxes as a `System.Int32`, and an INTEGER array automatically boxes as a `"System.Int32[]"`.

However, you can explicitly box the ABL value or array using a .NET mapped type other than the default match with the ABL built-in BOX function. This function accepts an ABL primitive value or array as input and, by default, returns a boxed .NET type according to the .NET default match for the ABL data type of the input value or array. In order to box the value using a mapped type other than the .NET default match, you can pass an ABL keyword as a string to the function that indicates the explicit .NET type you want to use. For example, an ABL DECIMAL value can represent both a .NET `System.Decimal` (the default match) and a `System.Double` (among other possible types). If you need to box the ABL DECIMAL as a .NET `System.Double`, you can explicitly indicate this to the BOX function. Similarly, if you need to box an ABL DECIMAL array as a `"System.Double[]"`, you can use the same indication.

Note: When you unbox a `System.Object` using the UNBOX function, you cannot similarly specify a particular ABL primitive or primitive array type as the result. ABL always unboxes any `System.Object` using the default matching ABL type.

Another case for which you must use the BOX function or the UNBOX function is when you pass parameters between compatible ABL primitive or array types and .NET object types in the parameters of ABL methods, procedures, and user-defined functions. ABL raises a compiler error if you try to pass these types to each other directly in ABL routine parameters. Appropriate use of the BOX function or UNBOX function allows this type of parameter passing to occur without a compile-time error. Note, again, that ABL **does** support the automatic boxing and unboxing of .NET objects in parameter passing for .NET method calls.

Similarly, three additional cases exist (other than the need for explicit boxing) where you must specify the .NET data type mapping you want for a given ABL primitive type:

- When a method parameter is overloaded by multiple implicit .NET data type mappings for a passed ABL primitive type. You must specify the exact .NET data type when you pass the parameter to the method.
- When you override a method inherited from a .NET class, or when you implement (or override) a .NET interface (or abstract) method, property, or event, and the types of any associated parameters, properties, or return values are .NET mapped data types. You must specify the exact .NET data type in the definition of each .NET mapped parameter, property, and return value.
- When you pass an ABL primitive value (**not** an array) to a .NET method or constructor parameter that is a `System.Object`, and you want the result to be a .NET mapped type other than the default match. You must indicate the explicit mapped type on the passed ABL primitive argument. A common use case is the `SetValue( )` method of the `System.Array` class, which sets the value of a .NET array element. If the .NET type of the array element is other than the default match, you must indicate the .NET mapped type for the value parameter to match the array definition.

- When you reference a constructed .NET generic type (described further in this entry) using type parameters that include a .NET mapped type.

To indicate a non-default .NET mapped type in the previous cases where you want an explicit mapped type to be used, the syntax for the following ABL elements allows you to specify an appropriate ABL keyword:

- BOX function
- DEFINE PROPERTY statement
- METHOD statement
- Parameter definition syntax
- Parameter passing syntax

This keyword is referred to as an *AS data type*, because you specify it for a passed parameter using the AS option. So, for example, when you override a .NET method, you must explicitly specify .NET data type for each .NET mapped parameter, property, or return type. If the .NET data type is a default match, you must simply use the matching ABL data type. Otherwise, you must indicate the appropriate AS data type keyword for the data type of the method parameter, return type, or property definition.

Table 25: Explicit mappings between ABL and .NET data types lists each explicit .NET data type mapping for a given ABL primitive type. For each listed .NET data type, you indicate this explicit mapping either by using the corresponding ABL primitive type (for a default match) or by using the appropriate option to specify the AS data type that corresponds to the explicit .NET data type you want to map. For more information on specifying the AS data type option when using the BOX function or when calling overloaded .NET methods, see the reference entries for the BOX function and Parameter passing syntax in this book. For more information on specifying AS data types when overriding a .NET method, or when implementing (or overriding) a .NET interface (or abstract) method, property, or event, see the METHOD statement, the DEFINE PROPERTY statement, the DEFINE EVENT statement, or the Parameter definition syntax reference entry, as appropriate.

Note: The AS data types in Table 25: Explicit mappings between ABL and .NET data types represent some different data types than you can specify using the AS option to pass a COM method parameter. For more information on passing COM method parameters, see Syntax for accessing COM object properties and methods.

Table 25: Explicit mappings between ABL and .NET data types

Explicit .NET object type	Explicit C# primitive type	ABL primitive type	ABL AS data type
`System.Boolean`	`bool`	LOGICAL[27]	–
`System.Byte`	`byte`	INTEGER	UNSIGNED-BYTE[28]
`System.SByte`	`sbyte`	INTEGER	BYTE
`System.DateTime`	N/A	DATETIME[27]	–
`System.Decimal`	`decimal`	DECIMAL[27]	–
`System.Int16`	`short`	INTEGER	SHORT

[27] The explicit .NET data type is the default match for the corresponding listed ABL primitive type. To specify a .NET default matching data type, simply pass the ABL primitive value as an overloaded method parameter or BOX function parameter, or define the overridden or implemented .NET method parameter, return type, or property as the corresponding ABL primitive type.

[28] To disambiguate overloaded methods, you cannot pass an ABL array and use an AS data type on an array parameter of a .NET method call. You must pass the argument as the .NET-defined array object type instead.

Explicit .NET object type	Explicit C# primitive type	ABL primitive type	ABL AS data type
`System.UInt16`	`ushort`	INTEGER	UNSIGNED-SHORT
`System.Int32`	`int`	INTEGER[27]	–
`System.UInt32`	`uint`	INT64	UNSIGNED-INTEGER
`System.Int64`	`long`	INT64[27]	–
`System.UInt64`	`ulong`	DECIMAL	UNSIGNED-INT64
`System.Double`	`double`	DECIMAL	DOUBLE
`System.Single`	`float`	DECIMAL	FLOAT
`System.Char`	`char`	CHARACTER	SINGLE-CHARACTER
`System.String`	`string`	CHARACTER/ LONGCHAR[27]	–

A *.NET generic type* is a class or interface defined so that it functions as one of several different types, depending on how you reference its type name. A reference to a .NET generic type name includes one or more type parameters, each of which specifies a data type that the generic type can use in its implementation. When you reference the generic type name in ABL, you substitute a specific data type for each type parameter defined for the generic type. This reference then identifies the generic type as a *constructed type*. The notation for a generic type that you see in .NET documentation or in a class browser, where the type parameters are not resolved, is called an *open type*. An open type reference contains only placeholders for the parameters in the type name, such as <T>, which defines the single parameter for the following generic type:

```
System.Collections.Generic.List<T>
```

In ABL, you can only reference a .NET generic type as a constructed type using a type name that has the following syntax:

```
"namespace.object-name<type-parameter[ , type-parameter ]...>"
```

The *namespace* is a .NET namespace and the syntax from *object-name* up to and including the right angle bracket (>) forms the *dotNET-object-name* as described in the Type-name syntax reference entry. The left and right angle brackets (<>) are the part of .NET generic type name references that enclose the type parameters, as shown in the previous example. Each *type-parameter* in the parameter list represents a placeholder for a specific .NET data type. The number of type parameters and the data type that you can specify for each *type-parameter* in a constructed type reference depends on the generic type definition. However, you can never specify an ABL object type or an ABL-extended .NET class type as the data type of any *type-parameter*; it can only be a pure .NET type. The quotes are required in order to allow for the angle brackets and any spaces in the type name.

The definition for each *type-parameter* in a .NET generic type definition can specify constraints that determine the .NET types you can substitute for a given parameter when you reference the constructed type. If these constraints on a *type-parameter* allow you to specify one or more .NET mapped types, you must specify an appropriate explicit mapping for each such type when you specify the *type-parameter* in ABL, as described in Table 25: Explicit mappings between ABL and .NET data types.

For example, to define an object reference to a `System.Collections.Generic.List<T>` that is constructed as a list of `System.Int16`, you might use one of the following ABL statements:

```
VAR CLASS System.Collections.Generic.List<SHORT> shortList.

DEFINE VARIABLE shortList AS
  CLASS System.Collections.Generic.List<SHORT> NO-UNDO.
```

You can also reference an array of a generic type and define generic types with a type parameter that is an array. For more information, see the information on .NET arrays in the notes of this reference entry.

You can use a .NET generic type in all the same contexts as any other .NET type except to define an ABL class that:

- Inherits from a .NET generic class
- Implements a .NET generic interface

Also, while you can cast an object reference to a .NET generic type using the CAST function, you cannot cast to a .NET generic type using the DYNAMIC-CAST function.

For more information on how to identify .NET generic types and understand the constraints on their type parameters, see the .NET documentation on MSDN. For more information on working with .NET generic types in ABL, see *Use GUI for .NET Objects in ABL Applications*.

Notes

- ABL provides built-in data types, built-in class data types, and user-defined class data types. References to built-in data types follow these rules:
  - Like most other keywords, references to specific built-in data types appear in all UPPERCASE, using a font that is appropriate to the context. No uppercase reference ever includes or implies any data type other than itself.
  - Wherever *integer* appears, this is a reference to the INTEGER or INT64 data type.
  - Wherever *character* appears, this is a reference to the CHARACTER, LONGCHAR, or CLOB data type.
  - Wherever *decimal* appears, this is a reference to the DECIMAL data type.
  - Wherever *numeric* appears, this is a reference to the INTEGER, INT64, or DECIMAL data type.

  References to built-in class data types appear in mixed case with initial caps, for example, `Progress.Lang.Object`. References to user-defined class data types appear in mixed case, as defined for a given application example.
- INT64 support applies to all of the ABL built-in methods and functions that take integer-expression parameters. That is, integer-expression parameters can be either INT64 expressions or INTEGER expressions.
- All intermediate calculations are carried out in 64-bit arithmetic. For example, 2,000,000,000 * 100 / 100 gives the correct result whether the target field is INTEGER or INT64. However, although 2,000,000,000 * 100 does not cause an overflow, you must assign the result to an INT64 field. If you assign it to an INTEGER field, the AVM generates a run-time error.

- When you copy one MEMPTR (M1) to another MEMPTR (M2), the data that M1 points to is also copied. Therefore, MEMPTR M1 points to memory location L1, and MEMPTR M2 points to memory location L2 which contains a copy of the data in L1. You must change the data in both memory locations if you want both MEMPTRs to reflect the change. To clear memory after using the MEMPTRs, you must execute SET-SIZE = 0 on both MEMPTRs to be sure that both memory locations are cleared.
- Since RAW variables are limited in size to 32K and MEMPTR variables can be up to 2GB - 1 byte in size, if a MEMPTR with a size greater than 32K is copied to a RAW variable, the AVM generates an error.
- Both a primitive type or object type can be defined as an array. Use the EXTENT option (for the DEFINE VARIABLE statement) or brackets (for the VAR statement) when defining or creating a field, variable, or object to establish an array. For example:

```
DEFINE VARIABLE someIntArray AS INTEGER EXTENT 4.

VAR INTEGER[4] someIntArray.
```

  The variable is now defined as an array of four integers. Since the size is set to 4, this is a *determinate* array. You can also define an *indeterminate* array by omitting the constant integer value after EXTENT (in the DEFINE VARIABLE statement) or within the brackets (for the VAR statement). In this case, the number of elements in the array is undefined.

  To refer to an individual element in an array, enclose the INTEGER index (subscript) using bracket syntax. This is known as a *subscripted* array reference. For example:

```
someIntArray[2] = 128.
```

  Here, 2 references the second element in the INTEGER array.

  Arrays can also be manipulated as a whole for array-to-array deep copy operations and to pass or return parameters. By omitting the brackets, a reference to the field, variable, or object name is a reference to the entire array. This is called an *unsubscripted* array reference. For example:

```
someIntArray = anotherIntArray.
```

  Here, each element of the `anotherIntArray` will be copied into the corresponding element of the `someIntArray`. This is called a *deep copy*. Note that unsubscripted array references are not supported in expressions or comparison operations. For more information on array assignments, see the Assignment (=) statement reference entry.

- ABL supports access to the following kinds of .NET object types:
  - **Classes** — Viewed and managed like ABL classes with support for additional features that are unique to .NET classes, such as inner classes and indexers for indexed properties. For information on accessing instances of .NET classes, see the Class-based object reference entry.
  - **Interfaces** — Viewed and managed as ABL interfaces with support for additional features that are unique to .NET, such as inner interfaces.
  - **Structures** — Viewed and managed similar to ABL classes. Structure types are supported using syntax native to each .NET language, for example, using the struct keyword in C# and C++. For information on accessing instances of .NET structures, see the Class-based object reference entry. The essential difference between .NET structures and most other .NET classes is that structures inherit from `System.ValueType` and are therefore value types. Thus, all structures are passed within .NET, and between ABL and .NET, by value. However within ABL, structure objects are passed, like all other class

instances, by reference. Therefore, when you access a structure from .NET, you reference a copy of the object in ABL that is separate from the object in .NET, and when you pass an ABL reference to a structure back to .NET, .NET gets a copy of the object that is separate from the object that is referenced in ABL. Structures therefore have different object management requirements in ABL than reference type objects. For more information, see the information on ABL support for value types in *Use GUI for .NET Objects in ABL Applications*.

- **Enumerations** — Like ABL enumerations, .NET enumerations are classes that correspond to a named set of constant values with a single underlying data type. Each of these constant values corresponds to a member of a given enumeration class. Each .NET language allows you to define and reference enumeration members using its own syntax. ABL also provides syntax that allows you to reference .NET enumerations as object types. For more information, see the Enumeration member access reference entry. Enumerations inherit from the `System.Enum` structure, which inherits from `System.ValueType`. Thus, like structures, enumerations are value types that are passed by value between .NET and ABL. However, unlike .NET languages that can view enumerations as values, ABL views enumerations only as objects that are passed by reference, like any other class instance. Enumerations therefore have similar object management requirements to structures in ABL. For more information, see the information on ABL support for value types in *Use GUI for .NET Objects in ABL Applications*.

You can instantiate .NET class or structure instances using the NEW function (classes), as with any ABL class. However, you cannot create an enumeration object. ABL can only reference enumeration objects that have already been defined in .NET. For information on specific .NET object types, see the documentation provided by the vendor for that object type.

- ABL supports widening relationships between certain ABL data types. Widening allows you to pass an argument to a method parameter that has a different data type than the parameter, depending on the data flow (INPUT or OUTPUT). Thus, the target of the data flow can be a different data type if it can hold the largest value provided by the source of the data flow. When passing .NET method parameters or getting and setting .NET property values, ABL supports additional widening relationships between the ABL data type being passed and the .NET data type of the parameter. For more information, see the description of widening for .NET parameters in the Parameter passing syntax reference entry.
- A .NET array is an object that extends the `System.Array` class. You can access a .NET array in ABL using an object reference, like any other object. Thus, in ABL, you can access all .NET arrays, of all dimensions, whose element type is either an ABL-supported .NET object type (such as `System.Windows.Forms.Form`) or a .NET mapped data type (such as `System.Int16` or C# `short`). You can also create .NET arrays directly in ABL by creating instances of the `System.Array` class. Note that while you cannot explicitly define a variable as a `System.Int16`, ABL does allow you to define a .NET array where the element type is a `System.Int16` (or any other mapped object type). .NET arrays have the following class hierarchy in ABL, in order of derivation from the ABL root class:

  1. `Progress.Lang.Object`
  2. `System.Object`
  3. `System.Array`
  4. Any array class of a specified element type, for example, `"System.Int32[]"`, which specifies a one-dimensional array of `System.Int32` elements

  You can define references to .NET arrays or ABL arrays (extents) of .NET array references:

```
DEFINE VARIABLE buttonArray AS
  CLASS "System.Windows.Forms.Button[]" NO-UNDO.

DEFINE VARIABLE buttonArrayExt AS
  CLASS "System.Windows.Forms.Button[]" EXTENT 3 NO-UNDO.
```

You can also define references to arrays of a generic type or to generic types that have type parameters that are arrays. For example:

```
DEFINE VARIABLE shortListArrayObj AS
  CLASS "System.Collections.Generic.List<SHORT>[]" NO-UNDO.

DEFINE VARIABLE shortListArrayExt AS
  CLASS "System.Collections.Generic.List<SHORT>" EXTENT 3 NO-UNDO.

DEFINE VARIABLE buttonArrayList AS CLASS
   "System.Collections.Generic.List<System.Windows.Forms.Button[]>"
   NO-UNDO.

DEFINE VARIABLE buttonArrayListExt AS CLASS
   "System.Collections.Generic.List<System.Windows.Forms.Button[]>"
   EXTENT 3 NO-UNDO.
```

Because all .NET array objects inherit from `System.Array`, you can access any .NET array object using the members of the `System.Array` class. To help create .NET array objects, OpenEdge provides a Progress.Util.TypeHelper class to specify the `System.Type` object needed for creating .NET array objects. For more information on working with .NET arrays, see the sections on accessing .NET arrays in *Use GUI for .NET Objects in ABL Applications*.

- If a .NET array is a multi-dimensional array, you can only work with it as a .NET array object, using the `System.Array` access mechanisms. However, if a .NET array is a one-dimensional array, you can work with it in two different ways:
  - Directly as a .NET array object
  - As an ABL array by directly assigning the .NET array to an equivalent ABL array, working with the resulting ABL array using ABL mechanisms, and directly assigning the reworked ABL array back to a .NET array
- An array assignment can occur between .NET and ABL arrays of compatible element types, either by direct assignment of one array to another using the Assignment (=) statement or the ASSIGN statement, or by passing array parameters to .NET methods using Parameter passing syntax. In these specific cases, ABL performs automatic boxing and unboxing between the compatible ABL and .NET array types. In general, you can assign an ABL or .NET array of .NET value types (such as `System.Drawing.Size`) only to another ABL or .NET array of identical value type elements. ABL makes an exception if the ABL array in the assignment is an array of primitive type elements (such as INTEGER), in which case ABL allows assignment to or from a compatible .NET array of mapped types (such as `"System.Int32[]"` or `"System.Byte[]"`). Otherwise (for reference types), elements of the target array must be identical to or higher in the class hierarchy than the elements of the source array. For example, you can assign an array of `System.Windows.Forms.Form` elements to an array of `System.Object` elements.
- While ABL does automatically box and unbox entire arrays for supported array assignments, ABL does **not** automatically box and unbox the **elements** of the source and target arrays. So, for example, you cannot assign an ABL array of INTEGER elements to a .NET array of `System.Object` elements (`"System.Object[]"`).
- Given that array element types are compatible, how an array assignment works, depends on the system context (ABL or .NET) of the arrays involved. In general, if either or both the target or source of the assignment is an ABL array, this results in a deep copy of all the array elements from the source array to the target. If both the target and source of the assignment is a .NET array, as with any object assignment, this results in an object reference copy, where the target references the same array as the source. Other array interactions depend on the array type of the target.

Caution: Because assignment between a .NET array and an ABL array requires a deep copy, note the performance impact it might have on your application before coding this operation. For a .NET array, you might prefer to work directly with the object reference, and access individual elements using `System.Array` mechanisms.

- For a .NET and ABL array assignment where the target is an ABL array:
  - The source array must be another ABL array, a one-dimensional .NET array, or a `System.Object` whose type at run time is a one-dimensional .NET array, and the element data type of the target array must be compatible with the element data type of the source, or the assignment raises an error. In general, the compatibility between source and target element types follows standard ABL rules for data type assignments, both within ABL and between ABL and .NET. For more information on data type compatibility for assignment and parameter passing, see the Assignment (=) statement and the Parameter passing syntax reference entry.
  - If the target is an ABL array of `System.Object` elements and the source array is an array of .NET mapped types (such as `System.Int32`) or ABL primitive types, ABL raises a compile-time error, because (as noted previously) ABL does not automatically do the unboxing and boxing operations that are required on the elements of each array.
  - If the target has an indeterminate EXTENT, the elements from the source array are copied to the target, setting its EXTENT to the number of elements in the source array.
  - If the target has a set EXTENT and the number of elements in the source array match the target EXTENT, the elements from the source array are copied to the target; otherwise, the assignment raises an error. This is true for both ABL and .NET source arrays.
- For a .NET and ABL array assignment where the target is a .NET array object or other object reference:
  - The source array can be any ABL array or .NET array with an element data type that is compatible with the target. If the element data type of the target array is a .NET value type, a .NET source array must be defined with elements of an identical value type. If the target array is a .NET array of mapped types, an ABL source array must be defined with elements of an ABL primitive type that implicitly maps to the target element type (see Table 24: Implicit mappings between .NET and ABL data types. If the target element type is not mapped (such as `System.Drawing.Size`), an ABL source array (like a .NET array) must be defined with elements of an identical value type. Otherwise, assignment to arrays of .NET reference type elements follow standard rules for assigning object references of related class and interface types.
  - If the target is a `System.Array` or a `System.Object`, you can assign to it any .NET source array object or any ABL source array that is defined as a supported .NET object type or as an ABL primitive type that maps implicitly to a .NET mapped data type (see Table 24: Implicit mappings between .NET and ABL data types). If the source is a .NET array object, ABL simply assigns it to the `System.Array` or `System.Object` reference. If the source is an ABL array, ABL creates a new .NET array object to hold the ABL array elements and assigns it to the `System.Array` or `System.Object` reference. In addition, if the ABL array elements have a primitive type, ABL automatically maps the ABL array elements into the default matching .NET object type before storing them in the specified `System.Array` or `System.Object`.
  - If the target is a .NET array of `System.Object` elements (`"System.Object[]"`), you can also assign any compatible ABL or .NET array to it. Note (as previously described) that a source array with .NET value type elements (such as `System.Drawing.Size` or ABL INTEGER, which resolves to the value-type, `System.Int32`) is not compatible. .NET requires that the element types must be identical in array assignments involving value type elements.
  - If the target is a `Progress.Lang.Object`, you can assign to it any .NET source array object, but not a native ABL array (which is not an object).

- The EXTENT of any ABL source array and the dimensions and size of a .NET source array do not matter. As noted previously, ABL copies the elements of an ABL source array to a newly created .NET array object and stores the object into the target object reference.

  Caution: The index for ABL arrays is 1-based, while the index for .NET arrays is generally 0-based.

- You can pass the Unknown value (?) as a parameter to a .NET method or assign the Unknown value (?) to a .NET property or data member. ABL translates the Unknown value (?) in these cases to the .NET null value. For the numeric and logical .NET primitive types listed in Table 24: Implicit mappings between .NET and ABL data types, when they are set to null, .NET sets a different default value—0, 0.0, or no—depending on the data type. In these cases, ABL also returns the .NET null value as the ABL Unknown value (?).
- ABL does not do any mapping between `System.Data.DataSet` or `System.Data.DataTable` method parameters, properties, or data members on one hand and ABL ProDataSets and temp-tables on the other. ABL supports data binding between ProDataSets or temp-tables (among other data sources) and .NET form controls using the `Progress.Data.BindingSource` class (the ProBindingSource). For more information, see the Progress.Data.BindingSource class reference entry. However, you can always directly access .NET `DataSet` and `DataTable` objects as any other .NET object, using their class members.

See also

Assignment (=) statement, ASSIGN statement, BOX function, CLASS statement, INTERFACE statement, NEW function (classes), Progress.Data.BindingSource class, Type-name syntax, Progress.Util.TypeHelper class, UNBOX function

DATASERVERS function

Returns a list of database types your OpenEdge product supports from where it is executed. The DATASERVERS function takes no arguments.

Syntax

```
DATASERVERS
```

The DATASERVERS function returns a character string containing a comma-separated list of database types. For example:

```
"PROGRESS,MSS,ORACLE"
```

Which indicates licensed support for the OpenEdge database, DataServer for Microsoft SQL Server, and DataServer for Oracle.

You can use the returned string with the LOOKUP function to determine whether a particular type of database is supported.

Example

The following example displays a selection list of all supported database types:

r-dserv.p

```
DEFINE VARIABLE db-types AS CHARACTER NO-UNDO VIEW-AS SELECTION-LIST
  INNER-CHARS 20 INNER-LINES 3 LABEL "DataServers".

FORM db-types.
db-types:LIST-ITEMS = DATASERVERS.
UPDATE db-types.
```

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, LOOKUP function, NUM-DBS function, PDBNAME function, SDBNAME function

DATA-SOURCE-MODIFIED function

Returns TRUE if data in the data source associated with the specified ProDataSet temp-table buffer has been modified.

Syntax

```
DATA-SOURCE-MODIFIED( buffer-name )
```

buffer-name

The name of a ProDataSet temp-table buffer.

Notes

- The AVM sets the value of this function from the SAVE-ROW-CHANGES( ) method.
- The DATA-SOURCE-MODIFIED function corresponds to the DATA-SOURCE-MODIFIED attribute.
- You can invoke the DATA-SOURCE-MODIFIED function from within a WHERE clause (unlike the corresponding attribute).

DATE function

Converts a single character string, a set of month, day, and year values, an integer expression, a DATETIME expression, or a DATETIME-TZ expression into a DATE value.

If the DATE function cannot produce a valid date given the specified argument(s), it returns a run-time error.

Syntax

```
DATE ( month , day , year )
```

```
DATE ( string )
```

```
DATE ( integer-expression )
```

```
DATE ( datetime-expression )
```

month

A constant, field name, variable name, or expression whose value is an integer from 1 to 12, inclusive.

day

An expression whose value is an integer from 1 to the highest valid day of the *month*.

year

An expression whose value is the year (for example, 1994).

string

A character string containing a date value to convert into a DATE data type. The string value must have the format specified by the Date Format (-d) startup parameter (the default is mdy). Note that -d sets the display format, not the date storage format, which is fixed. Furthermore, date constants entered in procedures, or as initial values in the Data Dictionary, are always specified in month/day/year format.

You do not have to specify separator characters for the month, day, and year components of the date string; however, slashes(/), periods(.), and hyphens(-) are accepted as separator characters.

integer-expression

An expression that evaluates to a signed integer value that represents the number of days since the origin of the ABL date data type. Usually this integer is obtained from a previous operation where the date was converted to an integer using the INTEGER(*ABL-date*) or INT64(*ABL-date*) function.

The value of the expression cannot be a date value before 12/31/-32768 or after 12/31/32767.

Note: The resulting date from the DATE(*integer-expression*) function is guaranteed to be a valid ABL date only if the *integer-expression* originated from the INTEGER(*ABL-date*) or INT64(*ABL-date*) function.

---

datetime-expression

An expression that evaluates to a DATETIME or DATETIME-TZ. The DATE function returns the date portion of the *datetime-expression* as a DATE.

If *datetime-expression* is a DATETIME-TZ, the DATE function returns the date relative to the time zone of the DATETIME-TZ value. For example, a DATETIME-TZ field, *fdt*, is created in London (time zone UTC+00:00) with a value of *May 5, 2002 at 7:15:03.002 am*. DATE (*fdt*) returns *05/05/2002*, regardless of the session's time zone.

Examples

This procedure reads data from an input file that contains date information from another system stored as character strings without slashes or dashes between month, day, and year. It tries to convert these dates to ABL dates. Some formats cannot be successfully converted. For example:

r-date.p

```
/* r-date.p */
DEFINE VARIABLE cnum  AS CHARACTER NO-UNDO FORMAT "x(3)".
DEFINE VARIABLE cdate AS CHARACTER NO-UNDO FORMAT "x(16)".
DEFINE VARIABLE iday  AS INTEGER   NO-UNDO.
DEFINE VARIABLE imon  AS INTEGER   NO-UNDO.
DEFINE VARIABLE iyr   AS INTEGER   NO-UNDO.
DEFINE VARIABLE ddate AS DATE      NO-UNDO.

INPUT FROM VALUE(SEARCH("r-date.dat")).
REPEAT:
  SET cnum cdate.
  ASSIGN
    imon  = INTEGER(SUBSTR(cdate,1,2))
    iday  = INTEGER(SUBSTR(cdate,4,2))
    iyr   = INTEGER(SUBSTR(cdate,7,2))
    /* Works for years within 50 of 2000 */
    iyr   = iyr + (IF (iyr < 50) THEN 2000 ELSE 1900)
    ddate = DATE(imon,iday,iyr).

  DISPLAY ddate.
END.
INPUT CLOSE.
```

The following example shows the DATE (*string*) syntax:

r-date2.p

```
/* r-date2.p */
DEFINE VARIABLE cnum  AS CHARACTER NO-UNDO FORMAT "x(3)".
DEFINE VARIABLE cdate AS CHARACTER NO-UNDO FORMAT "x(16)".
DEFINE VARIABLE ddate AS DATE      NO-UNDO FORMAT "99/99/9999".

INPUT FROM VALUE(SEARCH("r-date.dat")).
REPEAT:
  SET cnum cdate.
  ddate = DATE(cdate).
  DISPLAY ddate.
END.
INPUT CLOSE.
```

This example produces the following output. It produces no date for the first example since spaces are not a valid date separator:

```
cnum  cdate            ddate
----  ---------------- ----------

 nyd  01 01 86
 ad   11-28-52         11/28/1952
 red  12/08/56         12/08/1956
 nsm  10.01.01         10/01/2001
 hrl  1/8/1994         10/08/1994
```

See also

ADD-INTERVAL function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, ETIME function, INTERVAL function, ISO-DATE function, MONTH function, MTIME function, NOW function, TIME function, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

DATETIME function

Converts date and time values, or a character string, into a DATETIME value.

Note: If any argument is the Unknown value (?), the result is the Unknown value (?).

Syntax

```
DATETIME ( date-exp [ , mtime-exp ] )
```

```
DATETIME ( string )
```

```
DATETIME ( month , day , year , hours , minutes
      [ , seconds [ , milliseconds ] ] )
```

date-exp

An expression whose value is a DATE.

mtime-exp

An expression whose value is an integer representing the number of milliseconds since midnight.

string

A character expression whose value is a string containing a DATETIME. The date portion of the string must have the format specified by the DATE-FORMAT attribute. The time portion must be in a valid time format (HH:MM:SS, and so on).

month

An expression whose value is an integer from 1 to 12, inclusive.

day

An expression whose value is an integer from 1 to the highest valid day of the month.

year

An expression that evaluates to a year.

hours

An expression whose value is an integer from 0 to 23, inclusive.

minutes

An expression whose value is an integer from 0 to 59, inclusive.

seconds

An expression whose value is an integer from 0 to 61, inclusive. The upper limit is 61 for leap seconds.

milliseconds

An expression whose value is an integer from 0 to 999, inclusive.

Example

Following is an example of using the DATETIME function:

```
DEFINE VARIABLE my-datetime AS DATETIME NO-UNDO.

/* This statement is equivalent to "my-datetime = NOW". */
my-datetime = DATETIME(TODAY, MTIME).
```

The following statements result in the same DATETIME value:

```
my-datetime = DATETIME(5, 5, 2002, 7, 15, 3).
my-datetime = DATETIME("05-05-2002 07:15:03").
```

See also

ADD-INTERVAL function, DATE function, DATE-FORMAT attribute, DATETIME-TZ function, DAY function, ETIME function, INTERVAL function, ISO-DATE function, MONTH function, MTIME function, NOW function, TIME function, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

DATETIME-TZ function

Converts a date, time, and time zone value, or a character string, into a DATETIME-TZ value.

Note: If any argument is the Unknown value (?), the result is the Unknown value (?).

Syntax

```
DATETIME-TZ ( date-exp [ , mtime-exp [ , timezone-exp ] ] )
```

```
DATETIME-TZ ( datetime-exp [ , timezone-exp ] )
```

```
DATETIME-TZ ( datetime-tz-exp [ , timezone-exp ] )
```

```
DATETIME-TZ ( month , day , year , hours , minutes
            [ , seconds [ , milliseconds [ , timezone-exp ] ] ] )
```

```
DATETIME-TZ ( string )
```

date-exp

An expression whose value is a DATE.

mtime-exp

An expression whose value is an integer representing the number of milliseconds since midnight.

timezone-exp

An expression whose value is an integer representing the time zone offset from Coordinated Universal Time (UTC) in minutes. If not specified the function uses the ABL session's time zone, as specified by the TIMEZONE attribute.

datetime-exp

An expression whose value is a DATETIME.

datetime-tz-exp

An expression whose value is a DATETIME-TZ. If the *timezone-exp* is not specified, then the *datetime-tz-exp* value is returned as is. If the *timezone-exp* is specified, then the function returns the *datetime-tz-exp* value in the *timezone-exp* time zone.

month

An expression whose value is an integer from 1 to 12, inclusive.

day

An expression whose value is an integer from 1 to the highest valid day of the month.

year

An expression that evaluates to a year.

hours

An expression whose value is an integer from 0 to 23, inclusive.

minutes

An expression whose value is an integer from 0 to 59, inclusive.

seconds

An expression whose value is an integer from 0 to 61, inclusive. The upper limit is 61 for leap seconds.

milliseconds

An expression whose value is an integer from 0 to 999, inclusive.

string

A character expression whose value is a string containing a DATETIME-TZ. The date portion of the string must have the format specified by the DATE-FORMAT attribute. The time portion must be in a valid time format (HH:MM:SS, and so on). If the string contains a time zone, it must be in +HH:MM format. If the string does not contain a time zone, the DATETIME-TZ inherits the time zone of the machine running the ABL session as specified by the TIMEZONE attribute.

Example

Following is an example of using the DATETIME-TZ function:

```
DEFINE VARIABLE my-datetime-tz AS DATETIME-TZ NO-UNDO.

/* This statement is equivalent to "my-datetime-tz = NOW". */
my-datetime-tz = DATETIME-TZ(TODAY, MTIME, TIMEZONE).
```

The following statements result in the same DATETIME-TZ value (when SESSION:DATE-FORMAT is mdy):

```
my-datetime-tz = DATETIME-TZ(5, 5, 2002, 7, 15, 3, 0, -300).
my-datetime-tz = DATETIME-TZ("05-05-2002 07:15:03-05:00").
```

See also

ADD-INTERVAL function, DATE function, DATE-FORMAT attribute, DATETIME function, DAY function, ETIME function, INTERVAL function, ISO-DATE function, MONTH function, MTIME function, NOW function, TIME function, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

DAY function

Evaluates a date expression and returns a day of the month as an INTEGER value from 1 to 31, inclusive.

Syntax

```
DAY ( date )
```

```
DAY ( datetime-expression )
```

date

An expression whose value is a DATE.

datetime-expression

An expression that evaluates to a DATETIME or DATETIME-TZ. The DAY function returns the day of the month of the date part of the DATETIME or DATETIME-TZ value.

Example

This procedure determines the date one year from a given date, allowing for leap years. You could simply determine a date 365 days later by adding 365 to the d1 variable, but that might not produce the correct result (for example, 1/1/92 + 365 days is 12/31/92).

r-day.p

```
DEFINE VARIABLE d1    AS DATE    NO-UNDO LABEL "Date".
DEFINE VARIABLE d2    AS DATE    NO-UNDO LABEL "Same date next year".
DEFINE VARIABLE d-day AS INTEGER NO-UNDO.
DEFINE VARIABLE d-mon AS INTEGER NO-UNDO.

REPEAT:
  SET d1.
  d-day = DAY(d1).
  d-mon = MONTH(d1).
  IF d-mon = 2 AND d-day = 29 THEN d-day = 28.
  d2 = DATE(d-mon,d-day,YEAR(d1) + 1).
  DISPLAY d2.
END.
```

See also

ADD-INTERVAL function, DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, ETIME function, INTERVAL function, ISO-DATE function, MONTH function, MTIME function, NOW function, TIME function, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

DBCODEPAGE function

Returns, as a character string, the name of a connected database's code page.

Syntax

```
DBCODEPAGE ( {integer-expression | logical-name | alias} )
```

integer-expression

The sequence number of a database the ABL session is connected to. For example, DBCODEPAGE(1) returns information on the first database the ABL session is connected to, DCODEPAGE(2) returns information on the second database the ABL session is connected to, etc. If you specify a sequence number that does not correspond to a database the ABL session is connected to, the DBCODEPAGE function returns the Unknown value (?).

logical-name

A character expression that specifies the database by its logical name or alias.

A code page maps each character in a character set to a numeric value. For an OpenEdge database, DBCODEPAGE returns the code page of the database represented by the integer expression, logical name, or alias. For a non-OpenEdge database, DBCODEPAGE returns the value originally inserted when the schema was created.

There are three possible types of non-OpenEdge code pages:

- Physical data source for the database
- Code page of a non-OpenEdge vendor library linked in with an OpenEdge dataserver executable (either dynamically or at compile time)
- Code page that is in the schema holder that is part of the create activity

If any parameter is invalid, it returns the Unknown value (?).

Example

This procedure displays the logical name and code page of all connected databases:

r-dbcp.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

REPEAT ix = 1 TO NUM-DBS:
  DISPLAY LDBNAME(ix) DBCODEPAGE(ix) FORMAT "x(19)".
END.
```

Notes

A database must be connected in order for the DBCODEPAGE function to work as described.

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, PDBNAME function, SDBNAME function

DBCOLLATION function

Returns, as a character string, the name of the collating sequence for character set information contained in the database. This name corresponds to the definition of the collating sequence contained in the `convmap.dat` file, which usually resides in the $DLC directory. If any parameter is invalid, DBCOLLATION returns the Unknown value (?).

Syntax

```
DBCOLLATION
  ( {integer-expression | logical-name | alias} )
```

integer-expression

The sequence number of a database the ABL session is connected to. For example, DBCOLLATION(1) returns information on the first database the ABL session is connected to, DBCOLLATION(2) returns information on the second database the ABL session is connected to, etc. If you specify a sequence number that does not correspond to a database the ABL session is connected to, the DBCOLLATION function returns the Unknown value (?).

logical-name or *alias*

A character expression that specifies the database by its logical name or alias.

Example

This procedure displays the logical name and collation of all connected databases:

r-dbcoll.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO
.
REPEAT ix = 1 TO NUM-DBS:
  DISPLAY LDBNAME(ix) DBCOLLATION(ix) FORMAT "x(19)".
END.
```

Notes

- OpenEdge and non-OpenEdge DataServers can evaluate the syntactical expression stated in a DBCOLLATION function. However, the methods used to process multiple byte code pages can differ based

on the actual server used. Keep this point in mind if the actual results you receive differ from the results you expected.

- A database must be connected in order for the DBCOLLATION function to work as described.

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBRESTRICTIONS function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

DBNAME function

Returns, as a character string, the name of the logical database currently in use or the name of your first connected database.

Syntax

```
DBNAME
```

Example

This portion of a procedure defines a header frame to hold a date, page number, database name, and user ID:

r-dbname.p

```
DEFINE VARIABLE pageno AS INTEGER NO-UNDO FORMAT "zzz9" INITIAL 1.

FORM HEADER "Date:" TO 10 TODAY
  "Page:" AT 65 pageno SKIP
  "Database:" TO 10 DBNAME FORMAT "x(60)" SKIP
  "Userid:" TO 10 USERID WITH NO-BOX NO-LABELS.
VIEW.
```

Notes

- The AVM returns the database name in the same form you used when you connected to the database. If you used a fully qualified pathname, the AVM returns the full directory pathname (such as `/usr/acctg/gl` on UNIX or `C:\acctg\gl` in Windows). If you used a name relative to your current working directory, then the AVM returns that name (for example, gl).
- Unless you define a format, the database name is displayed in a character field with the default format of x(8).
- A database must be connected in order for the DBNAME function to work as described.

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

DBPARAM function

Returns, as a character string, a comma-separated list of the parameters used to connect to the database.

Syntax

```
DBPARAM ( integer-expression | logical-name | alias )
```

integer-expression

The sequence number of a database the ABL session is connected to. For example, DBPARAM(1) returns information on the first database the ABL session is connected to, DBPARAM(2) returns information on the second database the ABL session is connected to, etc. If you specify a sequence number that does not correspond to a database the ABL session is connected to, the DBPARAM function returns the Unknown value (?).

logical-name or *alias*

These forms of the DBPARAM function require a character expression as a parameter. An unquoted character string is not permitted. If the parameter is an alias or the logical name of a connected database, then the AVM returns the comma-separated parameter list. Otherwise, it returns the Unknown value (?).

Notes

- A database must be connected for the DBPARAM function to work as described.
- If the CONNECT statement does not contain a -db (database) parameter, which is permissible, the string DBPARAM returns includes the -db parameter and the database name.
- If the CONNECT statement contains the -pf parameter, which refers to a parameter file, the string DBPARAM returns includes the parameters in the file without "-pf" or any reference to the file.
- If the CONNECT statement contains a userid and a password, the string DBPARAM returns includes only the userid.
- The database can connect through the CONNECT statement, the command line, or an auto-connection.

See also

DBCODEPAGE function, DBCOLLATION function, DBTYPE function, DBVERSION function

DB-REMOTE-HOST function

Returns a character string containing the IP address of the database connection. The IP address format is determined by the Internet Protocol used when the connection was established. A single input parameter identifies the database, which can be either the logical database name or database number. If the database is not connected using TCP/IP or the `-ipver IPv6` startup parameter is not used, the function returns the Unknown value (?).

Syntax

```
DB-REMOTE-HOST ( { logical-name | integer-expression } )
```

logical-name

A character expression specifying the logical name of a connected database.

integer-expression

The sequence number of a connected database. For example, `DB-REMOTE-HOST(1)` returns information on the first connected database, `DB-REMOTE-HOST(2)` returns information on the second connected database, and so on.

Note

For more information on the Internet Protocol (IP) version (`-ipver`) startup parameter, see *Startup Command and Parameter Reference*.

See also

REMOTE-HOST attribute

DBRESTRICTIONS function

Returns a character string that describes features that are not supported for this database. You can use this function with OpenEdge DataServers.

Syntax

```
DBRESTRICTIONS
  ( {integer-expression | logical-name | alias}
    [ , table-name ]
  )
```

integer-expression

The sequence number of a database the ABL session is connected to. For example, `DBRESTRICTIONS(1)` returns information on the first database the ABL session is connected to,

DBRESTRICTIONS(2) returns information on the second database the ABL session is connected to, and so on. If you specify a sequence number that does not correspond to a database the ABL session is connected to, the DBRESTRICTIONS function returns the Unknown value (?).

logical-name or *alias*

These forms of the DBRESTRICTIONS function require a character expression as a parameter. An unquoted character string is not permitted. If the parameter is an alias or the logical name of a connected database, then the AVM returns the database restrictions string. Otherwise, it returns the Unknown value (?).

table-name

A character expression equal to the name of a table in the specified database. An unquoted character string is not permitted. If the table name is valid, DBRESTRICTIONS returns the list of unsupported features for the specified table. Otherwise, it returns the Unknown value (?).

Example

This procedure displays the logical name and database restrictions of all connected databases:

r-dbrest.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

REPEAT ix = 1 to NUM-DBS:
  DISPLAY LDBNAME(ix) LABEL "Database"
    DBRESTRICTIONS(ix) FORMAT "x(40)" LABEL "Restrictions".
END.
```

Notes

- If you want to use the DBRESTRICTIONS function for a database, you must be connected to the database in the current ABL session.
- DBRESTRICTIONS returns a string. This string is a comma-separated list of keywords that represent features not supported by the specified database. The following table shows the possible keywords and their descriptions.

Table 26: DBRESTRICTIONS keyword values

Keyword	Description
AGGREGATE	Cannot use the AGGREGATE statement.
COUNT-OF	Cannot use the COUNT-OF function.
DEFAULT-AREAS	Cannot use default areas.
LAST	Cannot invoke the FIND LAST statement.
PREV	Cannot invoke the FIND PREV statement.
READ-ONLY	The database or table is available for read only.
RECID	Cannot use the RECID function.

Keyword	Description
SET-CURRENT-VALUE	Cannot set the current value of sequence generators.
SETUSERID	Cannot use the SETUSERID function.

For example, if the database is accessed through a manager that does not support FIND LAST and FIND PREV, then the DBRESTRICTIONS function returns the string LAST, PREV.

- The possible keyword values returned by DBRESTRICTIONS depends on the DataServer type. The following table shows the possible values returned for each DataServer.

Table 27: DBRESTRICTIONS return values by DataServer

DataServer	Possible return values
ORACLE	"LAST,PREV,READ-ONLY,RECID,SETUSERID, SET-CURRENT-VALUE,DEFAULT-AREAS,AGGREGATE"
OpenEdge	"READ-ONLY,DEFAULT-AREAS,AGGREGATE"

Note: The available DataServers depend on your version of OpenEdge. For more information, see your OpenEdge DataServer documentation.

- The form of the returned string makes it easy to use with the ENTRY and LOOKUP function.
- If you connect to a database with the Read Only (-RO) parameter, the AVM lists the character string READ-ONLY in the restrictions list for that database.

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

DBTASKID function

Returns an INTEGER value that uniquely identifies a database's transaction.

Syntax

```
DBTASKID ( integer-expression | logical-name | alias )
```

integer-expression

The sequence number of a database the ABL session is connected to. For example, DBTASKID(1) returns information on the first database the ABL session is connected to, DBTASKID(2) returns information on the second database the ABL session is connected to, etc. If you specify a sequence

number that does not correspond to a database the ABL session is connected to, the DBTASKID function returns the Unknown value (?).

logical-name or *alias*

A character expression that evaluates to the logical name or alias of a database that is connected to the current ABL session. If the character expression does not evaluate to the logical name or alias of a connected database, DBTASKID returns the Unknown value (?).

Note: You must enclose all character strings in quotes.

Notes

- If the application is not in a transaction, DBTASKID returns the Unknown value (?).
- If the client is connected to two databases and both databases participate in the transaction, DBTASKID does not necessarily return the same value for each database. The value DBTASKID returns for a database is for that database only.
- DBTASKID returns the Unknown value (?) for DataServers and the temp-table database.
- DBTASKID is designed for database replication. When you create a log record for a transaction, you can call DBTASKID and store the transaction ID. When you load the transaction, you can group log records by transaction ID. For more information on database replication, see *Manage the OpenEdge Database*, and the reference entry for the RAW-TRANSFER statement.

See also

DBCODEPAGE function, DBCOLLATION function, DBTYPE function, DBVERSION function, LDBNAME function

DBTYPE function

Returns, as a character string, the database type of a currently connected database. This function returns one of the following strings: "MSS", "ORACLE", or "PROGRESS".

Syntax

```
DBTYPE ( integer-expression | logical-name | alias )
```

integer-expression

The sequence number of a database the ABL session is connected to. For example, DBTYPE(1) returns information on the first database the ABL session is connected to, DBTYPE(2) returns information on the second database the ABL session is connected to, etc. If you specify a sequence number that does not correspond to a database the ABL session is connected to, the DBTYPE function returns the Unknown value (?).

logical-name or *alias*

These forms of the DBTYPE function require a quoted character string or a character expression as a parameter. An unquoted character string is not permitted. If the parameter is an alias of a connected

database or the logical name of a connected database, then the AVM returns the database type. Otherwise, it returns the Unknown value (?).

Example

This procedure displays the logical name and database type of all connected databases:

r-dbtype.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

REPEAT ix = 1 TO NUM-DBS:
  DISPLAY LDBNAME(ix) DBTYPE(ix) FORMAT "x(40)".
END.
```

Note

You can reference the DBTYPE function within a preprocessor expression. For more information, see the &IF, &THEN, &ELSEIF, &ELSE, and &ENDIF preprocessor directives reference entry.

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

---

DBVERSION function

Returns, as a character string, the version number of an OpenEdge database.

Syntax

```
DBVERSION ( integer-expression | logical-name | alias )
```

integer-expression

The sequence number of a database the ABL session is connected to. For example, DBVERSION(1) returns information on the first database the ABL session is connected to, DBVERSION(2) returns information on the second database the ABL session is connected to, etc. If you specify a sequence number that does not correspond to a database the ABL session is connected to, the DBVERSION function returns the Unknown value (?).

logical-name or *alias*

These forms of the DBVERSION function require a quoted character string or a character expression as a parameter. If the parameter is an alias of a connected database or the logical name of a connected database, then the AVM returns the version number. Otherwise, it returns the Unknown value (?).

Example

This procedure displays the version number of all connected databases:

r-dbvers.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

REPEAT ix = 1 TO NUM-DBS:
   DISPLAY LDBNAME(ix) DBVERSION(ix) WITH 1 DOWN.
END.
```

Note

DBVERSION does not apply to non-OpenEdge data sources.

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

---

DDE ADVISE statement
(Windows only)

Instructs the dynamic data exchange (DDE) server associated with a conversation to either create or remove an advise link to the specified data item.

This statement is supported only for backward compatibility. Use the Component Object Model (COM) instead.

Note: Does not apply to SpeedScript programming.

Syntax

```
DDE ADVISE ddeid { START | STOP } ITEM name
   [ TIME seconds ]
   [ NO-ERROR ]
```

ddeid

An integer expression equal to the channel number of the conversation opened for the specified data item. It is the value returned by the DDE INITIATE statement that opened the conversation.

START

Instructs the server to create a link to a data item, and notify the ABL client when the specified data item changes value.

`STOP`

Instructs the server to remove the link to the specified data item, and stop monitoring its value.

`ITEM` *name*

Specifies the name of the server data item to which the link is created or removed. The data item *name* is a character expression that identifies the data item according to the conventions of the server application (for example, the row and column coordinates of a worksheet cell, such as R2C1 in Microsoft Excel). After creating a link, when the value of the data item specified by *name* changes, the AVM triggers a `DDE-NOTIFY` event for the frame that owns the conversation, allowing the client to retrieve the new value.

`TIME` *seconds*

Specifies the maximum number of seconds that the ABL client waits for the `DDE ADVISE` statement to complete, where *seconds* is an integer expression. If you do not specify the `TIME` option or specify a value of 0, the AVM waits indefinitely for the statement to complete.

`NO-ERROR`

The NO-ERROR option is used to prevent the statement from raising `ERROR` and displaying error messages.

Regardless of whether `NO-ERROR` is used, the AVM posts the error to the DDE frame `DDE-ERROR` attribute.

Example

The following fragment shows how to use the `DDE ADVISE` to set up a procedure to capture a rate-of-change value as it changes in a dynamic model run in a Microsoft Excel worksheet. The example assumes that the Microsoft Excel application is running, and has opened the default Excel worksheet, Sheet1, which runs the model.

After the conversation is opened, the DDE ADVISE statement links to the worksheet cell that maintains the latest rate-of-change value (second column of the fourth row, or R4C2). Every time this cell changes value, the AVM posts a DDE-NOTIFY event to the frame DDEframe, where the value is retrieved using the DDE GET statement, and stored as a decimal in the ChangeRate variable. Meanwhile, if the REPEAT block detects a ChangeRate value greater than 7.5%, the link to cell R4C2 is closed and the procedure continues.

```
DEFINE VARIABLE Sheet1     AS INTEGER.  NO-UNDO. /* DDE-ID to worksheet */
DEFINE VARIABLE ChangeRate AS DECIMAL   NO-UNDO  /* Rate of change... */
  INITIAL 0.0.                                   /* ...starting at zero */
DEFINE VARIABLE CellData   AS CHARACTER NO-UNDO. /* Worksheet cell output */
DEFINE VARIABLE DDEframe   AS HANDLE    NO-UNDO. /* DDE frame handle */

CREATE FRAME DDEframe
TRIGGERS:           /* DDE frame and code to receive  */
  ON DDE-NOTIFY DO: /* Rate of change data from Excel */
    DDE GET Sheet1 TARGET CellData ITEM "R4C2".
    ChangeRate = DECIMAL(CellData).
  END.
END TRIGGERS.
. . .
/* Open conversation with "Sheet1" and link to rate-of-change value. */
DDE INITIATE Sheet1 FRAME DDEframe APPLICATION "Excel" TOPIC "Sheet1".
DDE ADVISE Sheet1 START ITEM "R4C2".

/* Do some processing while the rate-of-change is within 7.5% */
REPEAT WHILE ChangeRate <= 7.5:
. . .
END. /* 7.5% processing */

/* Go on to other things once the rate of change goes above 7.5%. */
DDE ADVISE Sheet1 STOP ITEM "R4C2".
. . .
```

Notes

- After a DDE-NOTIFY event is triggered for the conversation, the client application must use the DDE GET statement in a trigger block for the event to retrieve the latest value for *name*.
- For more information on using the DDE protocol to exchange data with non-ABL applications, see *OpenEdge Programming Interfaces*.

See also

DDE GET statement, DDE INITIATE statement, NO-ERROR option

DDE EXECUTE statement
(Windows only)

Instructs a dynamic data exchange (DDE) server application to execute one or more application commands.

This statement is supported only for backward compatibility. Use the Component Object Model (COM) instead.

Note: Does not apply to SpeedScript programming.

Syntax

```
DDE EXECUTE ddeid COMMAND string
       [ TIME seconds ]
       [ NO-ERROR ]
```

ddeid

An integer expression equal to the channel number of a conversation opened to execute the specified command string. It is the value returned by the DDE INITIATE statement that opened the conversation. You can usually execute commands using a conversation opened for the System topic of the server application.

COMMAND *string*

Specifies the command or commands for the server to execute, where *string* is a character expression containing commands that are defined by the server application (for example, the [select(...)] command in Microsoft Excel).

TIME *seconds*

Specifies the maximum number of seconds that the ABL client waits for the DDE EXECUTE statement to complete, where *seconds* is an integer expression. If you do not specify the TIME option or specify a value of 0, the AVM waits indefinitely for the statement to complete.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Regardless of whether NO-ERROR is used, the AVM posts the error to the DDE frame DDE-ERROR attribute.

Example

The following fragment shows how to use the DDE EXECUTE statement. The procedure executes Microsoft Excel internally and opens a conversation for the Excel System topic. The System topic lets you execute Excel functions. This example uses the DDE EXECUTE statement to create a new Excel worksheet using the Excel new function:

```
DEFINE VARIABLE Sys      AS INTEGER NO-UNDO. /* DDE-ID to System topic */
DEFINE VARIABLE DDEframe AS HANDLE  NO-UNDO. /* DDE frame handle */
CREATE FRAME DDEframe.                       /* Create DDE frame */

/* DLL routine to execute an MS-Windows application. */
PROCEDURE WinExec EXTERNAL "kernel32.dll":
  DEFINE INPUT PARAMETER ProgramName AS CHARACTER.
  DEFINE INPUT PARAMETER Presentation AS LONG.
END PROCEDURE. /* WinExec */
. . .
/* Start Excel, open a DDE conversation with the Excel System topic, and create
   a worksheet. */
RUN WinExec (INPUT "Excel /e", INPUT 2). /* 1=normal, 2=minimized */
DDE INITIATE Sys FRAME DDEframe APPLICATION "Excel" TOPIC "System".
DDE EXECUTE Sys COMMAND "[new(1)]".
. . .
```

Notes

- For more information on commands available in your server application, see the documentation for that application.
- For more information on using the DDE protocol to exchange data with non-ABL applications, see *OpenEdge Programming Interfaces*.

See also

DDE INITIATE statement, NO-ERROR option

DDE GET statement
(Windows only)

Retrieves the value of a dynamic data exchange (DDE) server data item that has changed and triggered a `DDE-NOTIFY` event.

This statement is supported only for backward compatibility. Use the Component Object Model (COM) instead.

Note: Does not apply to SpeedScript programming.

Syntax

```
DDE GET ddeid TARGET field ITEM name
   [ TIME seconds ]
   [ NO-ERROR ]
```

ddeid

An integer expression that specifies the channel number of the conversation that triggered the `DDE-NOTIFY` event. You can obtain the value of *ddeid* from the `DDE-ID` attribute of the frame to which the `DDE-NOTIFY` event was posted.

`TARGET` *field*

Specifies a character field or variable that receives the value of the server data item as a character string.

`ITEM` *name*

Specifies the server data item that changed and triggered the `DDE-NOTIFY` event, where *name* is a character expression that identifies the name of the data item in the server application. You can obtain the value of *name* from the `DDE-ITEM` attribute of the frame to which the `DDE-NOTIFY` event was posted.

`TIME` *seconds*

Specifies the maximum number of seconds that the ABL client waits for the `DDE GET` statement to complete where *seconds* is an integer expression. If you do not specify the `TIME` option or specify a value of 0, the AVM waits indefinitely for the statement to complete.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Regardless of whether NO-ERROR is used, the AVM posts the error to the DDE frame DDE-ERROR attribute.

Example

The following fragment shows how to use the DDE GET statement to set up a procedure to capture a rate-of-change value as it changes in a dynamic model run in a Microsoft Excel worksheet. The example assumes that the Microsoft Excel application is running, and has opened the default Excel worksheet, Sheet1, which runs the model.

After the conversation is opened, the DDE ADVISE statement links to the worksheet cell that maintains the latest rate-of-change value (2nd column of the 4th row, or R4C2). Every time this cell changes value, the AVM posts a DDE-NOTIFY event to the frame DDEframe, where the value is retrieved using the DDE GET statement, and stored as a decimal in the ChangeRate variable. Meanwhile, if the REPEAT block detects a ChangeRate value greater than 7.5%, the the link to cell R4C2 is closed and the procedure continues.

```
DEFINE VARIABLE Sheet1     AS INTEGER NO-UNDO.   /* DDE-ID to worksheet */
DEFINE VARIABLE ChangeRate AS DECIMAL NO-UNDO    /* Rate of change... */
  INITIAL 0.0.                                   /* ...starting at 0 */
DEFINE VARIABLE CellData   AS CHARACTER NO-UNDO. /* Worksheet cell output */
DEFINE VARIABLE DDEframe   AS HANDLE    NO-UNDO. /* DDE frame handle */

CREATE FRAME DDEframe
TRIGGERS:           /* DDE frame and code to receive  */
  ON DDE-NOTIFY DO: /* Rate of change data from Excel */
    DDE GET Sheet1 TARGET CellData ITEM "R4C2".
    ChangeRate = DECIMAL(CellData).
  END.
END TRIGGERS.
. . .
/* Open conversation with Sheet1 and link to rate-of-change value. */
DDE INITIATE Sheet1 FRAME DDEframe APPLICATION "Excel" TOPIC "Sheet1".
DDE ADVISE Sheet1 START ITEM "R4C2".

/* Do some processing while the rate-of-change is within 7.5% */
REPEAT WHILE ChangeRate <= 7.5:
. . .
END. /* 7.5% processing */

/* Go on to other things once the rate of change goes above 7.5%. */
DDE ADVISE Sheet1 STOP ITEM "R4C2".
. . .
```

Notes

- The AVM posts each DDE-NOTIFY event to the frame that owns the conversation opened for the linked data item.
- You can invoke this function in the trigger block for each frame that owns a conversation containing advise links. Only frames that own conversations linked to data items with the DDE ADVISE statement can receive DDE-NOTIFY events.
- For more information on using the DDE protocol to exchange data with non-ABL applications, see *OpenEdge Programming Interfaces*.

See also

DDE ADVISE statement, DDE INITIATE statement, NO-ERROR option

DDE INITIATE statement
(Windows only)

Opens a dynamic data exchange (DDE) client conversation for a specified DDE server application and topic, and associates the new conversation with an ABL frame. To identify the conversation, the statement returns an integer as a unique channel number for this conversation.

This statement is supported only for backward compatibility. Use the Component Object Model (COM) instead.

Note: Does not apply to SpeedScript programming.

Syntax

```
DDE INITIATE ddeid FRAME frame-handle
  APPLICATION server-name TOPIC topic-name [ NO-ERROR ]
```

ddeid-var

An integer variable or field that receives the channel number for the newly opened DDE conversation.

FRAME *frame-handle*

Specifies the handle of the frame that owns the conversation, where *frame-handle* is a HANDLE expression. A frame can own more than one conversation. The AVM records the status of the most recent conversation exchange in a set of DDE frame attributes. These attributes record the status of every dynamic data exchange, including advise exchanges (exchanges triggered by DDE-NOTIFY events). The DDE frame attributes include:

- **DDE-ERROR** — The DDE error code returned by the most recent exchange
- **DDE-ID** — The channel number of the conversation that had the most recent exchange
- **DDE-ITEM** — The name of the data item referenced by the most recent exchange
- **DDE-NAME** — The name of the server application in the most recent exchange
- **DDE-TOPIC** — The name of the topic of the most recent exchange

APPLICATION *server-name*

Specifies the name of the server application for the conversation, where *server-name* is a character expression. The value of *server-name* must be unique for each DDE server on your system. It is usually the filename of the server executable without the extension (for example, the name EXCEL in Microsoft Excel).

TOPIC *topic-name*

Specifies the name of the topic of the conversation, where *topic-name* is a character expression. The value of *topic-name* identifies a category defined by the server application. This is usually the name of a file or other container that includes one or more data items (for example, the name of a

worksheet, such as Sheet1 in Microsoft Excel). An ABL client can only exchange data with server data items included in the topic of an open conversation.

`NO-ERROR`

The NO-ERROR option is used to prevent the statement from raising `ERROR` and displaying error messages.

Regardless of whether `NO-ERROR` is used, the AVM posts the error to the DDE frame `DDE-ERROR` attribute.

Example

The following fragment shows a typical use of the `DDE INITIATE` statement. It assumes that the Microsoft Excel application is running, and has created the default Excel worksheet, Sheet1. It then uses the `DDE INITIATE` statement to open a conversation with Sheet1 as the topic. This allows the AVM to exchange data with the cells of the worksheet. In this example, the fragment assigns column headings to the top row of the first three columns in the worksheet:

```
DEFINE VARIABLE Sheet1   AS INTEGER NO-UNDO. /* DDE-ID to worksheet topic */
DEFINE VARIABLE DDEframe AS HANDLE  NO-UNDO. /* DDE frame handle */

/* Create DDE frame */
CREATE FRAME DDEframe
  ASSIGN visible = TRUE
         hidden  = TRUE.
. . .
/* Open a DDE conversation with Sheet1 and assign column headings. */
DDE INITIATE Sheet1 FRAME DDEframe APPLICATION "Excel" TOPIC "Sheet1".
DDE SEND Sheet1 SOURCE "Name"      ITEM "R1C1".
DDE SEND Sheet1 SOURCE "YTD Sales" ITEM "R1C2".
DDE SEND Sheet1 SOURCE "State"     ITEM "R1C3".
. . .
```

Notes

- The specified DDE server application must be running on the Windows desktop before you can invoke the `DDE INITIATE` statement.
- You can close a DDE conversation in three ways: use the `DDE TERMINATE` statement, leave the scope of the frame that owns the conversation, or terminate the server application or topic associated with the application.
- For more information on using the DDE protocol (including DDE frame attributes) to exchange data with non-ABL applications, see *OpenEdge Programming Interfaces*.

See also

DDE TERMINATE statement, NO-ERROR option

DDE REQUEST statement
(Windows only)

Retrieves the current value of a dynamic data exchange (DDE) server data item associated with the specified DDE conversation.

This statement is supported only for backward compatibility. Use the Component Object Model (COM) instead.

Note: Does not apply to SpeedScript programming.

Syntax

```
DDE REQUEST ddeid TARGET field ITEM name
   [ TIME seconds ]
   [ NO-ERROR ]
```

ddeid

An integer expression that equals the channel number of the conversation opened for the specified data item. It is the value returned by the DDE INITIATE statement that opened the conversation.

TARGET *field*

Specifies a character field or variable that receives the value of the data item as a character string.

ITEM *name*

Specifies the name of the server data item from which to retrieve a value. The data item *name* is a character expression that identifies the data item according to the conventions of the server application (for example, the row and column coordinates of a worksheet cell, such as R2C1 in Microsoft Excel).

TIME *seconds*

Specifies the maximum number of seconds that the ABL client waits for the DDE REQUEST statement to complete, where *seconds* is an integer expression. If you do not specify the TIME option or specify a value of 0, the AVM waits indefinitely for the statement to complete.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Regardless of whether NO-ERROR is used, the AVM posts the error to the DDE frame DDE-ERROR attribute.

Example

The following fragment shows a typical use of the DDE REQUEST statement. It assumes that the Microsoft Excel application is running, and has created the default Excel worksheet, Sheet1. It then uses the DDE INITIATE statement to open a conversation with Sheet1 as the topic. This allows the AVM to exchange data with the cells of the worksheet.

In this example, the fragment builds 10 new Customer records from data obtained from the first 4 columns in the worksheet using the DDE REQUEST statement. The data includes Customer name, year-to-date sales, state, and zip code. (The requests start from row 2, because row 1 contains column headings.)

```
DEFINE VARIABLE Rowi     AS INTEGER   NO-UNDO. /* Worksheet row counter */
DEFINE VARIABLE ItemName AS CHARACTER NO-UNDO. /* Item Name */
DEFINE VARIABLE CustName AS CHARACTER NO-UNDO. /* Customer name receptor */
DEFINE VARIABLE YTDsales AS CHARACTER NO-UNDO. /* YTD sales receptor */
DEFINE VARIABLE StateAbr AS CHARACTER NO-UNDO. /* State name receptor */
DEFINE VARIABLE ZipCode  AS CHARACTER NO-UNDO. /* Zip code receptor */
DEFINE VARIABLE Sheet1   AS INTEGER   NO-UNDO. /* DDE-ID to worksheet */
DEFINE VARIABLE DDEframe AS HANDLE    NO-UNDO. /* DDE frame handle */

CREATE FRAME DDEframe.                         /* Create DDE frame */
. . .
/* Open a DDE conversation with Sheet1 and create 10 Customer records from
   the data in four columns of the worksheet. */
DDE INITIATE Sheet1 FRAME DDEframe APPLICATION "Excel" TOPIC "Sheet1".

REPEAT Rowi = 2 TO 11:
  CREATE Customer.
  Customer.CustNum = Rowi - 1.
  ItemName = "R" + STRING(Rowi) + "C1".
  DDE REQUEST Sheet1 TARGET CustName ITEM ItemName.
  Customer.Name = CustName.
  ItemName = "R" + STRING(Rowi) + "C2".
  DDE REQUEST Sheet1 TARGET YTDsales ITEM ItemName.
  Customer.YtdSls = DECIMAL(YTDsales).
  ItemName = "R" + STRING(Rowi) + "C3".
  DDE REQUEST Sheet1 TARGET StateAbr ITEM ItemName.
  Customer.State = StateAbr.
  ItemName = "R" + STRING(Rowi) + "C4".
  DDE REQUEST Sheet1 TARGET ZipCode  ITEM ItemName.
  Customer.PostalCode = INTEGER(ZipCode).
  RELEASE Customer.
END.
. . .
```

Note

For more information on using the DDE protocol to exchange data with non-ABL applications, see *OpenEdge Programming Interfaces*.

See also

DDE INITIATE statement, NO-ERROR option

DDE SEND statement
(Windows only)

Sends a new value to a dynamic data exchange (DDE) server data item associated with the specified DDE conversation.

This statement is supported only for backward compatibility. Use the Component Object Model (COM) instead.

Note: Does not apply to SpeedScript programming.

Syntax

```
DDE SEND ddeid SOURCE data ITEM name
   [ TIME seconds ]
   [ NO-ERROR ]
```

ddeid

An integer expression that equals the channel number of the conversation opened for the specified data item. It is the value returned by the DDE INITIATE statement that opened the conversation.

SOURCE *data*

Specifies the new value for the server data item, where *data* is a character expression that renders the new value in a format acceptable to the data item.

ITEM *name*

Specifies the name of the server data item to receive the new value. The data item *name* is a character expression that identifies the data item according to the conventions of the server application (for example, the row and column coordinates of a worksheet cell, such as R2C1 in Microsoft Excel).

TIME *seconds*

Specifies the maximum number of seconds that the ABL client waits for the DDE SEND statement to complete, where *seconds* is an integer expression. If you do not specify the TIME option or specify a value of 0, the AVM waits indefinitely for the statement to complete.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Regardless of whether NO-ERROR is used, the AVM posts the error to the DDE frame DDE-ERROR attribute.

Example

The following fragment shows a typical use of the DDE SEND statement. It assumes that the Microsoft Excel application is running, and has created the default Excel worksheet, Sheet1. It then uses the DDE INITIATE statement to open a conversation with Sheet1 as the topic. This allows the AVM to exchange data with the cells of the worksheet. In this example, the fragment assigns column headings to the top row of the first three columns in the worksheet using the DDE SEND statement.

```
DEFINE VARIABLE Sheet1   AS INTEGER NO-UNDO. /* DDE-ID to worksheet topic */
DEFINE VARIABLE DDEframe AS HANDLE  NO-UNDO. /* DDE frame handle */

CREATE FRAME DDEframe.                       /* Create DDE frame */
. . .
/* Open a DDE conversation with Sheet1 and assign column headings. */
DDE INITIATE Sheet1 FRAME DDEframe APPLICATION "Excel" TOPIC "Sheet1".
DDE SEND Sheet1 SOURCE "Name"      ITEM "R1C1".
DDE SEND Sheet1 SOURCE "YTD Sales" ITEM "R1C2".
DDE SEND Sheet1 SOURCE "State"     ITEM "R1C3".
. . .
```

Note

For more information on using the DDE protocol to exchange data with non-ABL applications, see *OpenEdge Programming Interfaces*.

See also

DDE INITIATE statement, NO-ERROR option

DDE TERMINATE statement
(Windows only)

Closes the specified dynamic data exchange (DDE) conversation.

This statement is supported only for backward compatibility. Use the Component Object Model (COM) instead.

Note: Does not apply to SpeedScript programming.

Syntax

```
DDE TERMINATE ddeid [ NO-ERROR ]
```

ddeid

An integer expression that equals the channel number of an open conversation. It is the value returned by the DDE INITIATE statement that opened the conversation.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Regardless of whether NO-ERROR is used, the AVM posts the error to the DDE frame DDE-ERROR attribute.

Example

The following fragment shows a typical use of the DDE TERMINATE statement. It assumes that the Microsoft Excel application is running, and has created the default Excel worksheet, Sheet1. It then uses the DDE INITIATE statement to open a conversation with Sheet1 as the topic, returning the channel number of the conversation to the variable, Sheet1. After exchanging data with the worksheet, the example closes the conversation with Sheet1 using the DDE TERMINATE statement.

```
DEFINE VARIABLE Sheet1   AS INTEGER NO-UNDO. /* DDE-ID to worksheet topic */
DEFINE VARIABLE DDEframe AS HANDLE  NO-UNDO. /* DDE frame handle */

CREATE FRAME DDEframe.                       /* Create DDE frame */
. . .
/* Open a DDE conversation with "Sheet1" and assign column headings. */
DDE INITIATE Sheet1 FRAME DDEframe APPLICATION "Excel" TOPIC "Sheet1".
DDE SEND Sheet1 SOURCE "Name"      ITEM "R1C1".
DDE SEND Sheet1 SOURCE "YTD Sales" ITEM "R1C2".
DDE SEND Sheet1 SOURCE "State"     ITEM "R1C3".
. . .
DDE TERMINATE Sheet1.
```

Notes

- Before closing a DDE conversation, remove all advise links in the conversation using the DDE ADVISE statement.
- Closing this conversation makes *ddeid* unavailable for further exchanges, but any other conversations open to the same server are still available.
- For more information on using the DDE protocol to exchange data with non-ABL applications, see *OpenEdge Programming Interfaces*.

See also

DDE ADVISE statement, DDE INITIATE statement, NO-ERROR option

DECIMAL function

Converts an expression of any data type, with the exception of BLOB, CLOB, and RAW, to a DECIMAL value.

Syntax

```
DECIMAL ( expression )
```

expression

If *expression* is a CHARACTER, then it must be valid for conversion into a number. (For example, 1.67 is valid but 1.x3 is not valid.) If *expression* is LOGICAL, then the result is 0 if *expression* is FALSE and 1 if *expression* is TRUE. If *expression* is a DATE, then the result is the number of days from 1/1/4713 B.C. to that date. If the value of *expression* is the Unknown value (?), then the result is also the Unknown value (?).

Example

The example procedure lets the user enter new values for `CreditLimit` in a special form. If the user enters the letter a, the procedure uses the standard a credit of 5000; if the user enters b, the procedure uses a value of 2000; if the user presses **RETURN**, the procedure uses a value of 1000. Otherwise, the user can enter any value for `CreditLimit`. The DECIMAL function converts the value entered into a decimal value.

r-decml.p

```
DEFINE VARIABLE new-max AS CHARACTER NO-UNDO FORMAT "x(10)".

REPEAT:
  PROMPT-FOR Customer.CustNum WITH FRAME credit.
  FIND Customer USING Customer.CustNum.
  DISPLAY Customer.CustNum Customer.Name Customer.CreditLimit
    WITH FRAME credit DOWN.
  DISPLAY "Enter one of:" SKIP(1)
    "a = 5000" SKIP
    "b = 2000" SKIP
    "RETURN = 1000"
    "A dollar value"
    WITH FRAME vals COLUMN 60.
  SET new-max WITH FRAME credit.
       IF new-max = "a" THEN Customer.CreditLimit = 5000.
  ELSE IF new-max = "b" THEN Customer.CreditLimit = 2000.
  ELSE IF new-max > "0" AND new-max < "999,999.99" THEN
    Customer.CreditLimit = DECIMAL(new-max).
  ELSE Customer.CreditLimit = 1000.

  DISPLAY Customer.CreditLimit WITH FRAME credit.
END.
```

See also

INTEGER function, STRING function

DECRYPT function

Converts encrypted data (a binary byte stream) to its original source format, and returns a MEMPTR containing the decrypted data.

Note: You must use the same cryptographic algorithm, initialization vector, and encryption key values to encrypt and decrypt the same data instance.

Syntax

```
DECRYPT ( data-to-decrypt [ , encrypt-key [ , iv-value [ , algorithm ] ] ] )
```

data-to-decrypt

The encrypted data to decrypt. The value may be of type RAW or MEMPTR.

encrypt-key

An optional RAW expression that evaluates to the encryption key (a binary value) originally used to encrypt the specified data. If you specify the Unknown value (?), the current value of the SYMMETRIC-ENCRYPTION-KEY attribute is used. If the value of the SYMMETRIC-ENCRYPTION-KEY attribute is also the Unknown value (?), the AVM generates a run-time error.

The AVM compares the size of the specified encryption key to the key size specified by the cryptographic algorithm. If the key sizes are inconsistent, the AVM generates a run-time error.

You can use the GENERATE-PBE-KEY function to regenerate the same encryption key originally used to encrypt the specified data as long as you specify the same password string, hash algorithm, number of iterations, and salt value.

Note: Do not use the GENERATE-RANDOM-KEY function to generate this encryption key. The random key it generates will always be different than the key originally used to encrypt the specified data.

You are responsible for generating, storing, and transporting this value.

iv-value

An optional RAW expression that evaluates to the initialization vector value to use with the specified encryption key in the original encryption operation. If you specify the Unknown value (?), the current value of the SYMMETRIC-ENCRYPTION-IV attribute is used.

algorithm

An optional CHARACTER expression that evaluates to the name of the symmetric cryptographic algorithm originally used to encrypt the specified data instance. If you specify the Unknown value (?), the current value of the SYMMETRIC-ENCRYPTION-ALGORITHM attribute is used.

For a list the supported cryptographic algorithms, see the SYMMETRIC-SUPPORT attribute reference entry.

See also

ENCRYPT function, SECURITY-POLICY system handle

DEFINE BROWSE statement

Defines and creates either a read-only or updateable browse widget that is created at compile time for use in one or more procedures, or within a single class.

Note: Does not apply to SpeedScript programming.

Syntax

```
DEFINE { [ [ NEW ] SHARED ] | [ PRIVATE ] }
  BROWSE browse-name
  QUERY query-name
    [ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ] [ NO-WAIT ]
  DISPLAY
    { column-list | record [ EXCEPT field . . . ] }
    [ browse-enable-phrase ]
    { browse-options-phrase }
    [ CONTEXT-HELP-ID expression ]
    [ DROP-TARGET ]
    [ TOOLTIP tooltip ]
```

`NEW SHARED BROWSE` *browse-name*

Defines and identifies a browse widget that can be used by other procedures. When the procedure containing this statement ends, the browse is no longer available.

`SHARED BROWSE` *browse-name*

Defines and identifies a browse that was created in another procedure with the DEFINE NEW SHARED BROWSE statement.

[`PRIVATE` **]** `BROWSE` *browse-name*

Defines and identifies a browse widget as a class-scoped object. A class-scoped handle-based object is not a member of a class, but provides a resource that is privately available to the class definition similar to a non-shared data element in a procedure definition. The option to specify the PRIVATE access mode is provided for readability. You cannot specify PRIVATE when defining a browse widget as a data element in a method or procedure.

Note: This option is applicable only when defining a class-scoped browse widget in a class definition (`.cls`) file.

`BROWSE` *browse-name*

Defines and identifies a browse widget whose query you can access only within the current procedure, class, or method of a class.

`QUERY` *query-name*

The name of the query to browse. You must have previously defined or opened the query.

[`SHARE-LOCK` **|** `EXCLUSIVE-LOCK` **|** `NO-LOCK` **]**

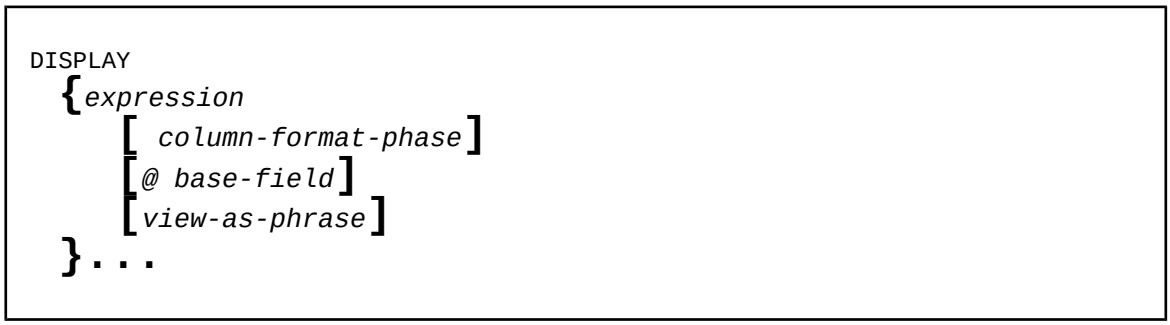
Specifies the locking mode for records retrieved by the browse widget. The default locking mode is NO-LOCK. To control locking during preselection for a query associated with a browse widget, use the SHARE-LOCK, EXCLUSIVE-LOCK, or NO-LOCK option in the OPEN QUERY statement that opens the query.

NO-WAIT

Specifies not to wait for a record that is currently locked by another process. Instead, the record in conflict will be made available in NO-LOCK mode and the LOCKED function for that record will return TRUE.

DISPLAY *column-list*

Specifies the column items to display in the browse. Note that the *column-list* cannot contain widgets other than fill-ins (default), combo-boxes, and toggle-boxes, and the *column-list* cannot contain SKIP options.

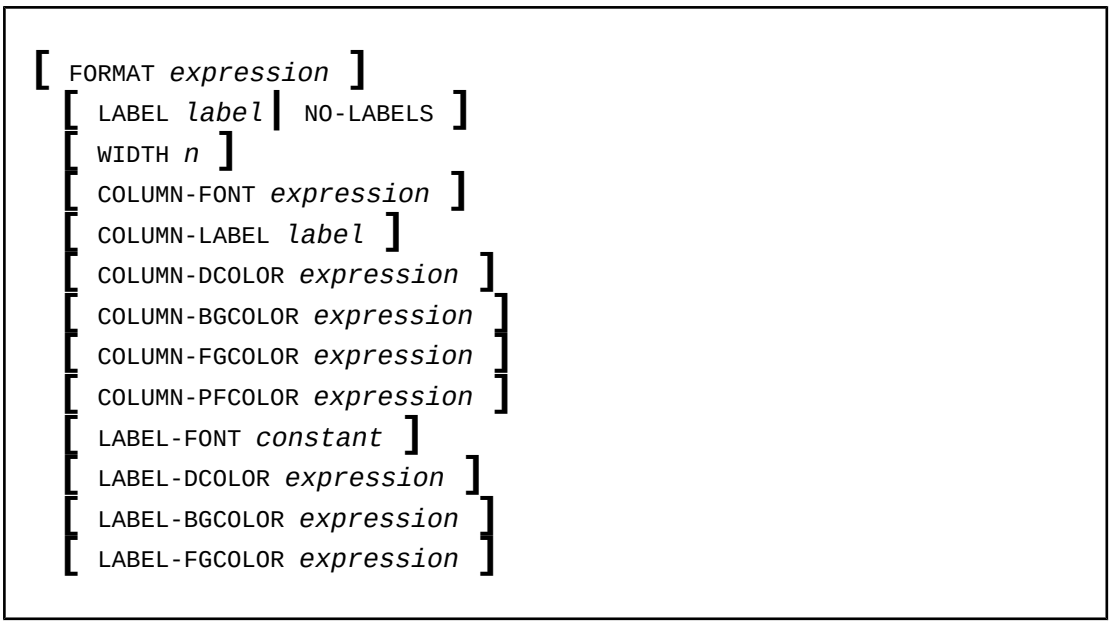


expression

A field name, variable, constant, or expression to display in each iteration of the browse frame.

column-format-phrase

Specifies the format for a value displayed in the browse. The *column-format-phrase* is a subset of the Format phrase.



WIDTH *n*

Specify a width for the browse column. *n* represents a multiplier of the average character width of the column font. Specifying a width smaller than the format string creates a scrolling browse cell, if the column is updateable.

For more information on FORMAT strings and label options, see the Format phrase reference entry. The column and label color and font options work like those specified in the *browse-options-phrase*. If color or fonts are specified with this phrase, they only affect the specific column and override similar options specified in the *browse-options-phrase*.

@base-field

The *base-field* must be the name of a field or variable; it cannot be an expression or constant.

The AVM reserves enough space for the *base-field* to hold the longest format displayed there. All right-justified fields (numeric fields that do not use side labels) are right justified within the reserved area.

To determine the format to use for displaying the expression at the *base-field*, the AVM looks at the following and uses the first format that applies:

- An explicit format used with the *expression.*
- If the expression is a character string constant, a format that accommodates that string.
- If the data type of the expression matches that of the *base-field*, the format of the *base-field.*
- The standard format of the expression as if it were displayed without a *base-field.*

view-as-phrase

Specifies the type of widget to use for displaying values in the browse column. This is the *view-as-phrase* syntax, which is a subset of the VIEW-AS phrase:

```
VIEW-AS combo-box-phrase | TOGGLE-BOX
```

If you do not specify a *view-as-phrase*, the widget type for the browse column is FILL-IN, by default.

Note: In a character interface, the browse column value is always displayed as a FILL-IN widget no matter what widget type you specify.

For more information, see the VIEW-AS phrase reference entry.

DISPLAY *record*

Specifies the record you want to display. If you specify a record, all fields from the record are displayed unless you use the EXCEPT option to eliminate specific fields.

See the Record phrase reference entry for more information.

EXCEPT *field* . . .

Specifies fields that are not displayed in the browse. You can use the EXCEPT option only if you specify a record name in the DISPLAY option.

browse-enable-phrase

Specifies which fields in the *column-list* are enabled for input.

```
ENABLE
  {{field[ HELP string]
     [ VALIDATE ( condition , msg-exp ) ]
     [ AUTO-RETURN ]
     [ DISABLE-AUTO-ZAP ]
    }...
    ALL [ EXCEPT field]
  }
```

List each field or variable from the *column-list* that you want enabled. Specify ALL to specify every item in the *column-list*. Use the EXCEPT option to exclude specific items when you use the ALL option.

For each field or variable, you can also specify custom help and validation, as shown in the next two entries.

HELP *string*

Represents a character string that you want to display whenever the user enters the frame field for the field or variable. When the user leaves the frame field, the AVM removes the help string from the message area. You must enclose the string in quotation marks ("").

VALIDATE (*condition*, *msg-expression*)

Specifies an expression against which you want to validate the data entered in a browse cell. The *condition* is a Boolean expression (a constant, field name, variable name, or expression) whose value is TRUE or FALSE.

When you use the VALIDATE option to validate a specific cell, any reference to that cell in *condition* is assumed to be the new input value. For example, in the *browse-enable phrase* below, the PromiseDate that is compared to the OrderDate is the new user input, not the existing data:

```
ENABLE PromiseDate VALIDATE(PromiseDate > OrderDate,
  "Promise date must be later than order date").
```

To validate a new value against another new value, use the INPUT qualifier, as shown:

```
ENABLE OrderDate PromiseDate VALIDATE(PromiseDate > INPUT OrderDate,
  "Promise date must be later than order date").
```

If the value of *condition* is FALSE, use *msg-expression* to display a specific message. You must enclose *msg-expression* in quotation marks ("").

The AVM processes validation criteria whenever the user attempts to leave the browse cell. If the cell value is not valid, the AVM displays *msg-expression* in the message area, causes the terminal to beep, and does not advance out of the browse cell.

If the user tabs to a cell, makes no changes, and leaves the cell, the AVM does not process the validation criteria specified with the VALIDATE option until you press **GO** (**F1**). If the user presses **ENDKEY** or **END-ERROR**, or an error occurs, the AVM does not test the validation criteria specified with the VALIDATE option.

If the input source for the procedure is a table, the AVM validates each input field (except those with a value of "-"). If the result of the validation is FALSE, *msg-expression* is displayed and the AVM treats the validation as an error.

To suppress the Data Dictionary validation criteria for a cell, use this VALIDATE option:

```
VALIDATE(TRUE,"")
```

AUTO-RETURN

Indicates whether the AVM behaves as if the user pressed the **RETURN** key when the user enters the last allowable character in a browse cell of the specified browse column.

DISABLE-AUTO-ZAP

Indicates whether the AVM should ignore the value of the browse column's AUTO-ZAP attribute and assume it is FALSE.

browse-options-phrase

Specifies options that affect the browse widget as a whole. The options affect both the layout and the function of the browse widget. Note that you cannot include aggregate-phrases (such as TOTAL, MIN, etc.) in this phrase. This is the syntax for *browse-options-phrase*:

```
WITH
  { [ constant ] DOWN [ WIDTH width ] | [ size-phrase ] }
  [ FGCOLOR expression ]
  [ BGCOLOR expression ]
  [ DCOLOR expression ]
  [ PFCOLOR expression ]
  [ LABEL-FONT expression ]
  [ LABEL-DCOLOR expression ]
  [ LABEL-FGCOLOR expression ]
  [ LABEL-BGCOLOR expression ]
  [ MULTIPLE | SINGLE ]
  [ SEPARATORS | NO-SEPARATORS ]
  [ NO-ASSIGN ]
  [ NO-ROW-MARKERS ]
  [ NO-LABELS ]
  [ NO-BOX ]
  [ FONT constant ]
  [ title-phrase ]
  [ NO-VALIDATE ]
  [ NO-SCROLLBAR-VERTICAL | SCROLLBAR-VERTICAL ]
  [ ROW-HEIGHT-CHARS | ROW-HEIGHT-PIXELS ] row-height
  [ FIT-LAST-COLUMN ]
  [ EXPANDABLE ]
  [ NO-EMPTY-SPACE ]
  [ DROP-TARGET ]
  [ NO-AUTO-VALIDATE ]
```

```
[ constant ] DOWN [ WIDTH width ]
```

The *constant* value is the number of rows displayed in the browse and must be at least 2. You can optionally specify the width of the browse, where *width* is the width of the browse in character units.

A *browse-options-phrase* must contain a DOWN option or a *size-phrase*.

size-phrase

Specifies the outer size of the browse border. When this option is used instead of the DOWN option, the AVM determines the number of rows that can be displayed in the browse. Following is the syntax for *size-phrase*:

```
{ SIZE SIZE-PIXELS }width BY height
```

For more information on *size-phrase*, see the SIZE phrase reference entry.

A *browse-options-phrase* must contain a DOWN option (optionally with a WIDTH option) or a *size-phrase*.

FGCOLOR *expression*

Specifies the foreground color for the browse in graphical environments, but not the label foreground color. The value of *expression* must be an integer value that specifies an entry in the color table. This option is ignored in character environments.

BGCOLOR *expression*

Specifies the background color for the browse in graphical environments. The value of *expression* must be an integer value that specifies an entry in the color table. This option is ignored in character environments.

DCOLOR *expression*

Specifies the display color for the browse in character environments. The value of *expression* must be an integer value that specifies an entry in the color table. This option is ignored in graphical environments.

PFCOLOR *expression*

Specifies the prompt color for the browse in character environments. The value of *expression* must be an integer value that specifies an entry in the color table. This option is ignored in graphical environments.

LABEL-FONT *constant*

Specifies the font of the browse labels.

LABEL-DCOLOR *expression*

Specifies the display color for the browse labels in character environments. The value of *expression* must be an integer value that specifies an entry in the color table. This option is ignored in graphical environments.

LABEL-FGCOLOR *expression*

Specifies the foreground color for the browse labels in graphical environments. The value of *expression* must be an integer value that specifies an entry in the color table. This option is ignored in character environments.

LABEL-BGCOLOR *expression*

Specifies the background color for the browse labels in graphical environments. The value of *expression* must be an integer value that specifies an entry in the color table. This option is ignored in character environments.

[MULTIPLE **|** SINGLE **]**

Specifies whether multiple rows can be selected from the browse or only a single row at one time. The default is SINGLE.

[SEPARATORS **|** NO-SEPARATORS **]**

Specifies whether row and column separators are displayed within the browse. The default is NO-SEPARATORS.

NO-ASSIGN

Disables automatic writes on new data in an updateable browse. If this option is not specified, data entered into an updateable browse is assigned on any action that results in a ROW-LEAVE event. This option is intended for use with user-defined triggers on the ROW-LEAVE event. Essentially, when this option is specified, you must make all data assignments by way of the updateable browse.

```
ON ROW-LEAVE OF my-browse DO:
  IF Customer.State:SCREEN-VALUE IN BROWSE my-browse NE "MA" THEN DO:
    MESSAGE "Customer is out of state."
    RETURN NO-APPLY.
  END.

  /* Your code. Transaction scope is up to you. */

  FIND CURRENT Customer.
  IF NOT CURRENT-CHANGED Customer THEN
    ASSIGN INPUT BROWSE field1 field2 field3 field4.
  ELSE MESSAGE "Record has changed since last read.".
END.
```

In the above example, the code looks for a special case where automatic database writes are not desirable and prevents them. The body of the trigger handles other processing before proceeding to commit the changes. First the trigger refinds the current Customer record and then uses the CURRENT-CHANGED function to see if it has changed while the user was updating the browse cells. If it has not changed, the changes are committed. If it has changed, the trigger would handle that condition, too.

Note that an ASSIGN statement with the INPUT BROWSE option can be mixed with other assignment types, as shown:

```
ASSIGN
  Name
  a = b
  INPUT FRAME my-frame c d
  INPUT BROWSE my-browse order-date promise-date
  INPUT e.
```

NO-ROW-MARKERS

By default, an updateable browse displays row markers, which allow the user to select currently displayed rows in an updateable browse widget. This option prevents row markers from being displayed.

NO-LABELS

Does not display labels above the columns of the browse.

NO-BOX

Does not display a box around the browse. If you do not use this option, the AVM displays a box around the data you are displaying.

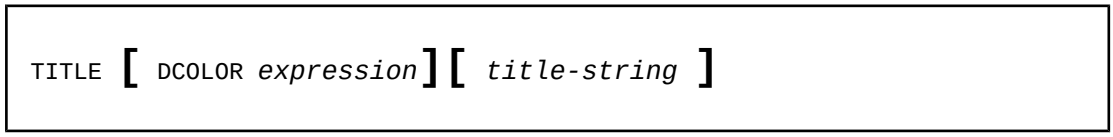
If you are sending data to a device other than a terminal and you do not use this option, the AVM omits the sides and bottom line of the box and replaces the top line with blanks.

FONT *constant*

Specifies the font of the browse. Labels also use this font, unless otherwise specified.

title-phrase

Displays a title as part of the top line of the box around the browse. For example:



The *title-string* is a constant, field name, variable name, or expression whose result is a character value. The *expression* is the value you want to display as a title. If *title-string* is a constant character string, it must be surrounded by quotes (""). The AVM automatically centers *title-string* in the top line of the browse box.

You can use the DCOLOR option to specify the color of the title under a character user interface.

You cannot specify the title color or font of a browse in a graphical interface.

NO-VALIDATE

Tells the AVM to ignore the validations conditions in the schema for all fields in the browse.

Since browses do not inherit the NO-VALIDATE option from a parent frame, if you want a browse to have this option, you must specify it explicitly.

NO-SCROLLBAR-VERTICAL | SCROLLBAR-VERTICAL

Indicates whether the browse displays a vertical scrollbar. The default is to display the vertical scrollbar.

ROW-HEIGHT-CHARS | ROW-HEIGHT-PIXELS *row-height*

An integer representing the browse's row height in either characters or pixels.

This option applies to graphical interfaces only.

FIT-LAST-COLUMN

Allows the browse to be displayed so that there is no empty space to the right and no horizontal scroll bar by potentially widening or shrinking the last browse column's width.

This option applies to graphical interfaces only.

When this option is specified, and the last browse column can be fully or partially displayed in the browse's viewport, then the last browse column's width is adjusted so that it fits within the viewport with no empty space to its right and no horizontal scroll bar.

If the last browse column is fully contained in the viewport with empty space to its right, it grows so that its right edge is adjacent to the vertical scroll bar.

If the last browse column extends outside the viewport, it shrinks so its right edge is adjacent to the vertical scroll bar and the horizontal scroll bar is not needed.

The default value is FALSE.

Note: The FIT-LAST-COLUMN option performs the same function as the EXPANDABLE option. Progress Software Corporation recommends that you use the FIT-LAST-COLUMN option instead of the EXPANDABLE option. This recommendation includes replacing EXPANDABLE with FIT-LAST-COLUMN in your current code.

EXPANDABLE

If you set a browse's EXPANDABLE option to TRUE, the AVM extends the right-most browse column horizontally to the browse's right edge, if necessary, to cover any white space that might appear there — unless you explicitly set the width of the right-most browse column using the WIDTH-CHARS or WIDTH-PIXELS option. The expansion of the right-most browse column might occur anytime the browse or another browse column is resized.

The right-most browse column expands only when there is no horizontal scroll bar. This is because when there is a horizontal scroll bar, no white space appears between the right edge of the right-most browse column and the right edge of the browse.

Note: The EXPANDABLE option performs the same function as the FIT-LAST-COLUMN option. Progress Software Corporation recommends that you use the FIT-LAST-COLUMN option instead of the EXPANDABLE option. This recommendation includes replacing EXPANDABLE with FIT-LAST-COLUMN in your current code.

NO-EMPTY-SPACE

Allows the browse to display with no empty space to the right and no horizontal scroll bar.

You cannot specify both NO-EMPTY-SPACE and FIT-LAST-COLUMN for the DEFINE BROWSE statement. If you specify both, the compiler displays an error message. If you set either the NO-EMPTY-SPACE option or the DEFINE BROWSE option to TRUE and one of them is already TRUE, a warning message displays at run time.

NO-AUTO-VALIDATE

Tells ABL to compile into the code all relevant validations it finds in the OpenEdge Data Dictionary, but to run the validations only when the code for a browse or a browse column specifically invokes the VALIDATE() method.

CONTEXT-HELP-ID *expression*

An integer value that specifies the identifier of the help topic for this browse in a help file specified at the session, window, or dialog box level using the CONTEXT-HELP-FILE attribute.

DROP-TARGET

Indicates whether the user can drop a file onto the object.

TOOLTIP *tooltip*

Allows you to define a help text message for a browse widget. The AVM automatically displays this text when the user pauses the mouse pointer over a browse widget for which a ToolTip is defined. You can add or change the TOOLTIP option at any time.

If TOOLTIP is set to "" or the Unknown value (?), then the ToolTip is removed from the browse. No ToolTip is the default. ToolTips are supported in Windows only.

Examples

This procedure sets up a read-only browse widget for the Customer table. The browse displays the CustNum and Name fields. A separate frame, `f2`, displays more information on the currently chosen Customer.

r-browse.p

```
DEFINE QUERY q1 FOR Customer.
DEFINE BROWSE b1 QUERY q1 DISPLAY CustNum Name
   WITH 17 DOWN TITLE "Customer Browse".

DEFINE FRAME f1 b1
  WITH SIDE-LABELS AT ROW 2 COLUMN 2.
DEFINE FRAME f2
  WITH 1 COLUMNS AT ROW 2 COLUMN 38.

ON VALUE-CHANGED OF b1 DO:
  DISPLAY Customer EXCEPT Customer.Comments WITH FRAME f2.
END.

OPEN QUERY q1 FOR EACH Customer.
ENABLE b1 WITH FRAME f1.
APPLY "VALUE-CHANGED" TO BROWSE b1.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

The VALUE-CHANGED event occurs each time the user selects a row within the browse widget. The associated database record is automatically placed into the record buffer. The trigger on the VALUE-CHANGED event displays that record in frame `f2`.

The APPLY statement causes the first Customer record to display before the user selects a record.

The second example sets up an updateable browse that displays some fields from the Customer table. Select a row marker to select a row. Select a cell to edit it. Select a column label to initiate a search. (The trigger on ROW-LEAVE is only necessary because the NO-ASSIGN option prevents automatic commitment of the data when the user leaves a row.)

r-brows2.p

```
DEFINE VARIABLE method-return AS LOGICAL NO-UNDO.

DEFINE QUERY q1 FOR Customer SCROLLING.
DEFINE BROWSE b1 QUERY q1 NO-LOCK DISPLAY Customer.CustNum Customer.Name
  Customer.Phone ENABLE Customer.Name Customer.Phone
  WITH 15 DOWN NO-ASSIGN SEPARATORS.
DEFINE BUTTON button1 LABEL "New Row".
DEFINE FRAME f1
  SKIP(1) SPACE(8) b1 SKIP(1) SPACE(8) button1
  WITH NO-BOX.

ON ROW-LEAVE OF b1 IN FRAME f1 DO: /* No-Assign Browser */
  /* If new row, create record and assign values in browse. */
  IF b1:NEW-ROW THEN DO:
    CREATE Customer.
    ASSIGN INPUT BROWSE b1 Customer.Name Customer.Phone.
    DISPLAY Customer.CustNum WITH BROWSE b1.
    method-return = b1:CREATE-RESULT-LIST-ENTRY().
    RETURN.
  END.
  /* If record exists and was changed in browse, update it. */
  IF BROWSE b1:CURRENT-ROW-MODIFIED THEN DO:
    GET CURRENT q1 EXCLUSIVE-LOCK.
    IF CURRENT-CHANGED Customer THEN DO:
      MESSAGE "This record has been changed by another user." SKIP
        "Please re-enter your changes.".
      DISPLAY Customer.CustNum Customer.Name Customer.Phone WITH BROWSE b1.
      RETURN NO-APPLY.
    END.
    ELSE /* Record is the same, so update it with exclusive-lock */
      ASSIGN INPUT BROWSE b1 Customer.Name Customer.Phone.
      /* Downgrade the lock to a no-lock. */
      GET CURRENT q1 NO-LOCK.
  END.
END.

ON CHOOSE OF button1 IN FRAME f1 DO: /* Insert */
  method-return = b1:INSERT-ROW("AFTER").
END.

OPEN QUERY q1 FOR EACH Customer.
ENABLE ALL WITH FRAME f1.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW
```

Notes

- You cannot define a SHARED or NEW SHARED browse widget in a persistent procedure. If you do, ABL raises ERROR on the RUN statement that creates the procedure.
- You cannot define a SHARED or NEW SHARED browse widget in a class definition (`.cls`) file. If you do, ABL generates a compilation error.
- The vertical scrollbar is displayed with the browse by default in Windows interfaces. It may be removed by setting the SCROLLBAR-VERTICAL attribute to FALSE or by specifying NO-SCROLLBAR-VERTICAL in the DEFINE BROWSE statement. If the horizontal scrollbar is needed, it is provided by default.
- The vertical scrollbar thumb size reflects the percentage of rows that are displayed in the viewport relative to the number of rows in the results list. If all the rows have not yet been read into the results list, the AVM uses the MAX-DATA-GUESS attribute to estimate the total size.
- You must put the browse into a frame on the same procedure level on which the browse is defined. For example, you cannot define a browse in an outer procedure and then display it in a frame defined within an internal procedure.

- You cannot display a browse widget in a down frame. The AVM automatically converts any frame containing a browse to a 1 down frame.
- You can modify the field values displayed in the current iteration of a browse by using the WITH BROWSE option of the DISPLAY statement:

```
DISPLAY { field | { value @ field } } . . .
  WITH BROWSE browse
```

- The browse widget has built-in support for the **HOME**, **END**, **PAGE-UP**, and **PAGE-DOWN** key functions.
- You can specify an application-defined widget ID for a compile-time defined browse widget using the *form-item* phrase in either the FORM statement or the DEFINE FRAME statement. See the FORM statement and DEFINE FRAME statement reference entries for more information.
- ABL treats the query associated with a browse as a scrolling query. You do not have to specify SCROLLING in the DEFINE QUERY statement.
- When you execute an OPEN QUERY or REPOSITION statement for the query associated with the browse, the browse is automatically adjusted to remain in sync with the query. However, when you execute a GET statement, the browse is not adjusted. You can use the GET statement to perform background processing without affecting the browse, but you must execute a REPOSITION statement to put the query and browse back in sync.
- The record locking behavior specified for a query in the DEFINE BROWSE statement overrides the record locking behavior specified with the OPEN QUERY statement. The default record locking behavior of a browse widget is NO-LOCK. The default record locking behavior of a query defined with the OPEN QUERY statement is SHARE-LOCK. If you define a query and a browse widget for the query without explicitly defining record locking behavior, the query will have the NO-LOCK behavior.
- For an updateable browse, the AVM re-gets the record with a SHARE-LOCK when the user first edits a row, if it initially has a NO-LOCK. The user then can make changes to the updateable cells in the row. When the user leaves a row with changes (moves to a new row or another widget), the AVM starts a transaction and gets the record with EXCLUSIVE-LOCK and NO-WAIT. If the AVM gets the record, the record is updated, the record is disconnected (removes the lock), the transaction ends, and the lock is downgraded to its original status. If the get record with EXCLUSIVE-LOCK fails, the transaction is backed out, an error message is displayed, and focus remains with the edited browse row with the changed data. To redisplay the original data, use the DISPLAY...WITH BROWSE statement.
- All LEAVE triggers for a browse row execute before the row changes are committed. If the LEAVE trigger returns a NO-APPLY, the changes are not committed.
- It is also possible to use an updateable browse to add new records and delete old ones.
- Browse widgets in Windows have user search capabilities. Special events allow you to extend these capabilities.
- To select more than one row in a browse widget, press **CTRL** and then click on the rows you want to select.
- When an updateable browse enters edit mode, all selected records are deselected. Essentially, a browse in edit mode ignores multiple selections.
- The ADD-CALC-COLUMN, ADD-COLUMNS-FROM and ADD-LIKE-COLUMN methods may be used on a compile-time defined browse to add a dynamic browse column. If a dynamic browse column in a compile-time defined browse is made updateable, the browse is changed to a NO-ASSIGN browse and you become responsible for any database update associated with it.
- The browse's QUERY attribute can now be set to the Unknown value (?). If this is done, all browse columns are removed.

- The browse's QUERY attribute can now be changed to any query. Previously, the new query had to have the same underlying fields as the original query. If the new query is different, all browse columns are removed. You must specify new columns with the ADD-CALC-COLUMN, ADD-COLUMNS-FROM, and ADD-LIKE-COLUMN methods.

See also

ADD-CALC-COLUMN( ) method, ADD-COLUMNS-FROM( ) method, ADD-LIKE-COLUMN( ) method, Class-based data member access, CLOSE QUERY statement, CREATE BROWSE statement, CURRENT-CHANGED function, CURRENT-RESULT-ROW function, DEFINE QUERY statement, DISPLAY statement, FIND statement, FORM statement, Format phrase, Frame phrase, GET statement, NUM-RESULTS function, OPEN QUERY statement, RUN statement

---

DEFINE BUFFER statement

ABL provides you with one default buffer for each table or temp-table that you use in a procedure or class. ABL uses that buffer to store one record at a time from the table as the records are needed during the procedure or class. If you need more than one record or buffer at a time for a table, you can use this statement to define alternate buffers that are created at compile time for use in one or more procedures, or within a single class or class hierarchy.

Syntax

```
DEFINE { [ [ NEW ] SHARED ] | [ PRIVATE | PROTECTED ] [ STATIC ] }
  BUFFER buffer-name
  FOR [ TEMP-TABLE ] table-name
  [ PRESELECT ] [ LABEL label-name ]
  [ NAMESPACE-URI namespace ] [ NAMESPACE-PREFIX prefix ]
  [ XML-NODE-NAME node-name ] [ SERIALIZE-NAME serialize-name ]
```

`NEW SHARED BUFFER` *buffer-name*

Defines and identifies a buffer that can be used by other procedures. When the procedure using this statement ends, the buffer is no longer available.

`SHARED BUFFER` *buffer-name*

Defines and identifies a buffer that was created in another procedure with the DEFINE NEW SHARED BUFFER statement.

`[ PRIVATE | PROTECTED ] [ STATIC ] BUFFER` *buffer-name*

Defines and identifies a buffer as a data member of a class, and optionally specifies an access mode (PRIVATE or PROTECTED) and scope (instance or STATIC) for that data member. You cannot specify these options when defining a buffer as a data element of a method (including constructors, destructors, and property accessors) or a procedure.

Note: The specified options are applicable only when defining a data member for a class in a class definition (`.cls`) file. The PRIVATE/PROTECTED modifier and the STATIC modifier can appear in either order, e.g., `DEFINE STATIC PRIVATE BUFFER myBuffer...` is valid syntax. Note also that you cannot shadow (override) the definition of a given buffer data member in a class hierarchy.

PRIVATE buffer data members can be accessed by the defining class. An instance can access the private buffer data members of another instance if both instances are of the same class. PROTECTED buffer data members can be accessed by the defining class and any of its derived classes. An instance can access protected buffer data members of a second instance that is at the same level or higher in the class hierarchy. The default access mode is PRIVATE. When you reference a buffer from another data member definition (such as a query) defined in the same class or class hierarchy, the access mode of the buffer cannot be more restrictive than the access mode of the referencing data member.

A buffer defined with the STATIC option is a static data member of the class type for which it is defined and is scoped to the ABL session where it is referenced. ABL creates one copy of the specified class static buffer on first reference to the class type, and ABL creates only one such copy for any number of instances of the class that you create. You can directly reference an accessible static buffer data member from any other static or instance class member defined in the same class or class hierarchy.

Without the STATIC option, ABL creates an instance buffer data member that is scoped to a single instance of the class where it is defined. ABL creates one copy of the specified instance buffer for each such class instance that you create. You cannot directly reference an instance buffer data member from a STATIC class member definition defined within the same class or class hierarchy.

Within a class, ABL defines the default buffer for any database table as a PRIVATE instance buffer. Thus, for example, you can only access the default buffer for the `Customer` table of the `Sports2020` database wherever a PRIVATE instance buffer can be accessed. Otherwise, you must define an alternate buffer data member for the table with an appropriate access mode and scope.

For more information on accessing buffers of different access modes and scopes, see the reference entry for Class-based data member access.

Note: Members of a class are grouped into six namespaces, including buffers/temp-tables, methods, variables/properties/events, ProDataSets, queries, and data-sources. Buffers and temp-tables defined as members of a class share the same namespace. There can be only one class member in this namespace with a given name.

For more information on where and how to define data members in a class, see the CLASS statement reference entry.

`BUFFER` *`buffer-name`*

Defines and identifies a buffer whose records you can access only within the current procedure, method of a class (including constructors, destructors, and property accessors), or as a PRIVATE data member of a class.

`FOR [ TEMP-TABLE ]` *`table-name`*

Identifies the name of the table for which you are defining an additional buffer. This can also be the built-in buffer name, `proc-text-buffer`, to define a buffer that returns table rows from a stored procedure.

To define a buffer for a table defined for multiple databases, you might have to qualify the table name with the database name. See the record definition in the Record phrase reference entry for more information.

Use the TEMP-TABLE option to define a buffer for a temp-table when the temp-table has the same name as a database table. Otherwise, ABL associates the buffer with the database table by default.

If you define the buffer as static, and *table-name* is the name of a temp-table, the temp-table must also be defined as a static member of a class.

PRESELECT

If you use the PRESELECT option with a DO or REPEAT block, the AVM creates an internal list of the records selected. The PRESELECT option tells the AVM to apply that internal list to the buffer you define. You can also use the PRESELECT option in the DEFINE SHARED BUFFER statement.

LABEL *label-name*

Specifies a label for the buffer. This label is used in error messages in place of the buffer name.

NAMESPACE-URI *namespace*

An optional CHARACTER constant that specifies the URI for the namespace of the buffer object.

NAMESPACE-PREFIX *prefix*

An optional CHARACTER constant that specifies the namespace prefix associated with the NAMESPACE-URI.

XML-NODE-NAME *node-name*

An optional CHARACTER constant that specifies the name of the XML element representing the temp-table buffer in an XML Document. The default is *buffer-name*. This option is useful when:

- reading a temp-table buffer from XML with a name that contains invalid characters for an ABL name or is an ABL keyword, or
- writing a temp-table buffer to XML with a name that is an invalid XML tag name.

Note: If you set SERIALIZE-NAME but do not set XML-NODE-NAME, the AVM sets XML-NODE-NAME equal to SERIALIZE-NAME.

SERIALIZE-NAME *serialize-name*

An optional CHARACTER constant that specifies the name of the element representing the temp-table buffer in an XML Document or JSON string. The default is *buffer-name*. This option is useful when:

- reading a temp-table buffer from XML or JSON with a name that contains invalid characters for an ABL name or is an ABL keyword, or
- writing a temp-table buffer to XML or JSON with a name that is an invalid XML tag name or an invalid JSON property name.

Note: If you set both XML-NODE-NAME and SERIALIZE-NAME, the READ-XML() and WRITE-XML() methods will use XML-NODE-NAME.

Examples

This procedure allows the user to create a new `Customer` record. Initially, the `City`, `State`, and `Country` fields are not shown. After the user enters a `PostalCode` value, the procedure searches for an existing Customer with the same postal code. If such a Customer is found, the `City`, `State`, and `Country` values from that record are displayed in the fields for the new record. The user can then update those fields.

r-defb.p

```
DEFINE BUFFER other-cust FOR Customer.

FORM Customer WITH FRAME cre-cust.

ON LEAVE OF Customer.PostalCode DO:
  FIND FIRST other-cust
    WHERE other-cust.PostalCode = Customer.PostalCode:SCREEN-VALUE
    AND other-cust.CustNum <> Customer.CustNum NO-ERROR.
  IF AVAILABLE other-cust THEN
    DISPLAY other-cust.City @ Customer.City
      other-cust.State @ Customer.State
      other-cust.Country @ Customer.Country
      WITH FRAME cre-cust.
  ENABLE Customer.City Customer.State Customer.Country WITH FRAME cre-cust.
END.

CREATE Customer.
UPDATE Customer EXCEPT Customer.City Customer.State Customer.Country
   WITH FRAME cre-cust.
```

The following gather a group of records so that the user can enter any table name and any set of record selection criteria and then look at the records in the table that meet those criteria:

r-defb2.p

```
DEFINE VARIABLE fname      AS CHARACTER NO-UNDO
  FORMAT "x(12)" LABEL "Table name".
DEFINE VARIABLE conditions AS CHARACTER NO-UNDO
  FORMAT "x(60)" LABEL "Conditions".

REPEAT:
 /* Get the name of a table and, optionally, some record selection criteria */
 UPDATE fname COLON 12 conditions COLON 12
   WITH SIDE-LABELS 1 DOWN.
 HIDE ALL.
 IF conditions <> "" THEN
   /* Pass the table name and the record selection criteria as parameters */
   RUN r-defb3.p fname "WHERE" conditions.
 ELSE
   RUN r-defb3.p fname.
END.
```

The `r-defb2.p` procedure gets the name of a table (such as Customer) and a condition (such as CreditLimit > 4000) and passes them as arguments to the `r-defb3.p` procedure:

r-defb3.p

```
DEFINE NEW SHARED BUFFER rec FOR {1} PRESELECT.

DEFINE VARIABLE flist AS CHARACTER NO-UNDO EXTENT 12.
DEFINE VARIABLE ix    AS INTEGER   NO-UNDO.

/* Look in _File for the table named in the fname variable */
FIND _File "{1}".

/* Store the table's field names in the first array */
FOR EACH _Field OF _File USE-INDEX _Field-Posit:
  IF i >= 12 THEN LEAVE.
  i = i + 1.
  flist[i] = _Field._Field-Name.
END.

/* Preselect records */
DO PRESELECT EACH rec {2} {3} {4} {5} {6} {7} {8} {9} {10} {11} {12}:
  /* Pass the filenames and all field names to r-defb4.p */
  RUN r-defb4.p "{1}" flist[1] flist[2] flist[3] flist[4] flist[5] flist[6]
    flist[7] flist[8] flist[9] flist[10] flist[11] flist[12].
END.
```

The `r-defb3.p` procedure:

- Lets you view the OpenEdge Data Dictionary. The _File table contains a record for each of your database tables.
- Lets you look up a record for a Customer table. For example, the user supplies Customer as a table name; the FIND statement in the `r-defb3.p` procedure translates to FIND _File Customer. The FIND statement finds, in _File, the record for the Customer table.
- Lets you view the _Field table in the OpenEdge Data Dictionary. The _Field table contains a single record for each of your database fields. The FOR EACH statement reads the name of each of those fields into the first array variable. If the table name is Customer, the first array variable contains the names of each of the fields in the Customer table.
- Lets you select records. For example, the user supplies the condition `CreditLimit` > 4000 in the table name. The DO PRESELECT EACH rec statement translates to DO PRESELECT EACH rec WHERE `MaxCredit` > 4000. The AVM goes through the Customer table and selects the records that meet the criteria. It creates a temp-table containing a pointer to each of those records. This list of preselected records is associated with the rec buffer.
- Runs `r-defb4.p`, passing the table name (Customer) and the names of all of the fields in that table.

The `r-defb4.p` procedure has access to the rec buffer (and through it to the set of preselected records). This connection is made by using PRESELECT on the DEFINE SHARED BUFFER statement. The `r-defb4.p` procedure displays those records.

r-defb4.p

```
DEFINE SHARED BUFFER rec FOR {1} PRESELECT.

REPEAT:
  FIND NEXT rec.
  DISPLAY {2} {3} {4} {5} {6} {7} {8} {9} {10} {11} {12} {13}
    WITH 1 COLUMN 1 DOWN.
END.
```

Because r-defb3.p and r-defb4.p use run-time argument passing, they cannot be precompiled. Having separate versions of r-defb4.p for each table and running the appropriate one in r-defb3.p, should improve response time. This approach is worthwhile if there are many lines of code in r-defb4.p a procedure.

If you define a NEW SHARED BUFFER in a procedure, then call a subprocedure that puts a record into that buffer, and display the buffer in the main procedure, the AVM displays this message:

```
Missing FOR, FIND or CREATE for Customer.
```

This message is displayed when the FIND statement is not in the main procedure:

```
/* Main procedure */

DEFINE NEW SHARED BUFFER x FOR Customer.

RUN proc2.p.
DISPLAY x.
```

```
/* proc2.p */

DEFINE SHARED BUFFER x FOR Customer.

FIND FIRST x.
```

To avoid this, explicitly scope the Customer record to the main procedure block. For example:

```
/* Main procedure */

DEFINE NEW SHARED BUFFER x FOR Customer.

RUN proc2.p.
DO FOR x:
  DISPLAY x.
END.
```

For examples of instance and static buffer data member definitions, see the descriptions of r-CustObj.cls, r-CustObjStatic.cls, and r-CustObjAbstract.cls in the CLASS statement reference entry.

Notes

- You cannot define a SHARED or NEW SHARED buffer in a class definition (.cls) file. If you do, ABL generates a compilation error.
- A buffer can be compile-time defined (often referred to as a *static* buffer object), where the buffer is defined and created at compile time using this statement, or it can be run-time defined (often referred to as a *dynamic* buffer object), where the buffer is defined and created at run time using the CREATE BUFFER statement and buffer object handle operations. A compile-time defined buffer can also be defined as a static data member of a class. In this case, it is a static buffer object that is also a class static data member.
- Every statement that uses a table name to refer to the default buffer can also use the name of a defined alternate buffer.
- All data definitions and field names are associated with a table, not a buffer. Data definitions and field names remain the same no matter what buffer you use.

- If two buffers contain the same record, a change to one of the buffers is automatically reflected in the other buffer.
- You can pass a buffer as a parameter to a procedure.
- A SHARED buffer remains in scope for an instance of a persistent procedure until the instance is deleted. This is true even if the original procedure that defined the buffer as NEW SHARED goes out of scope while the procedure instance remains persistent.

  If a trigger or internal procedure of a persistent procedure executes an external subprocedure that defines a SHARED buffer, ABL includes the persistent procedure in the resolution of the corresponding NEW SHARED buffer as though the procedure were on the procedure call stack.
- If you define a temp-table with the same name as a database table and you then define a buffer for that table name, by default the buffer will be associated with the database table, not with the temp-table. Use the TEMP-TABLE option to define a buffer for a temp-table when the temp-table has the same name as a database table.
- For more information on using the built-in buffer name `proc-text-buffer`, see the OpenEdge DataServer Guides (*Use the Microsoft SQL Data Server* and *Use the Oracle Data Server*).

See also

Class-based data member access, CREATE BUFFER statement, DEFINE PARAMETER statement, RUN statement, RUN STORED-PROCEDURE statement

DEFINE BUTTON statement

The DEFINE BUTTON statement defines a push button that is created at compile time for use within the current procedure or class.

Note: Does not apply to SpeedScript programming.

Syntax

```
DEFINE [ PRIVATE ] BUTTON button
   [ AUTO-GO | AUTO-ENDKEY ]
   [ DEFAULT ]
   [ BGCOLOR expression]
   [ CONTEXT-HELP-ID expression]
   [ DCOLOR expression]
   [ DROP-TARGET ]
   [ FGCOLOR expression]
   [ FONT number]
   [ IMAGE-DOWN image-phrase]
   [{ IMAGE | IMAGE-UP } image-phrase ]
   [ IMAGE-INSENSITIVE image-phrase]
   [ MOUSE-POINTER name]
   [ LABEL label]
   [ LIKE button]
   [ PFCOLOR expression]
   [ size-phrase ]
   [ NO-FOCUS [ FLAT-BUTTON ]]
   [ NO-CONVERT-3D-COLORS ]
   [ TOOLTIP tooltip]
  {[ trigger-phrase ]}
```

[PRIVATE] BUTTON *button*

Defines and identifies a button widget as a class-scoped object. A class-scoped handle-based object is not a member of a class, but provides a resource that is privately available to the class definition similar to a non-shared data element in a procedure definition. The option to specify the PRIVATE access mode is provided for readability. You cannot specify PRIVATE when defining a button widget as a data element in a method or procedure.

Note: This option is applicable only when defining a class-scoped button widget in a class definition (`.cls`) file.

BUTTON *button*

Defines and identifies a button widget for access only within the current procedure, class, or method of a class.

AUTO-END-KEY

Specifies that when you choose this button, the AVM applies the ENDKEY event to the frame.

AUTO-GO

Specifies that when you choose this button, the AVM applies the GO event to the frame.

DEFAULT

Specify DEFAULT to indicate that the button is a default button. A default button is one that handles all RETURN events when no other RETURN-enabling widget in the frame or dialog box has focus. RETURN-enabling widgets include any field-level widget for which a RETURN trigger is defined, or any button, whether or not it has a trigger defined. Thus, if a button has focus, that button handles the next RETURN event. If any other field-level widget without a RETURN trigger has focus, the default button handles the next RETURN event.

To make the button the default button for the frame in which it resides, you must also set the frame's DEFAULT-BUTTON option.

BGCOLOR *expression*

Supported only for backward compatibility.

CONTEXT-HELP-ID *expression*

An integer value that specifies the identifier of the help topic for this button in a help file specified at the session, window or dialog box level using the CONTEXT-HELP-FILE attribute.

DCOLOR *expression*

Specifies the display color for the button in character interfaces. This option is ignored in graphical interfaces.

FGCOLOR *expression*

Supported only for backward compatibility.

FONT *number*

Specifies the font for the button label. The value *number* must be an expression that resolves to an integer value. That integer must be associated with a specific font in your system environment files.

{ IMAGE **|** IMAGE-UP **}** *image-phrase*

An image that you want to appear within the button when the button is in its up state. If the image does not have a down state, for code readability you might want to use the IMAGE option instead of the IMAGE-UP option.

The IMAGE | IMAGE-UP *image-phrase* option is ignored in character interfaces.

The syntax of *image-phrase* is as follows:

```
FILE name
  [{ IMAGE-SIZE | IMAGE-SIZE-CHARS | IMAGE-SIZE-PIXELS }
     width BY height
  ]
  [ FROM {{ X n Y n}|{ ROW n COLUMN n}}]
```

For more information on this syntax, see the Image phrase reference entry.

IMAGE-DOWN *image-phrase*

An image that you want to appear within the button when the button is in its down state. The IMAGE-DOWN option is ignored in character interfaces.

For more information, see the Image phrase reference entry.

Note: The AVM draws the 3D effect only if a button has an up image, but no down image.

IMAGE-INSENSITIVE *image-phrase*

An image you want to appear within the button when the button is in its insensitive (disabled) state. This option is ignored in character interfaces.

For more information, see the Image phrase reference entry.

MOUSE-POINTER *name*

Specifies the mouse pointer for the button. The character value *name* is either the name of a predefined ABL pointer, or the name of a Windows `.cur` file that defines a pointer or an `.ani` file that contains an animated cursor.

LABEL *label*

The label displayed on the button. The name should describe the action invoked when the button is chosen. The value of *label* must be a string enclosed in quotes. The default label is the button name. If you use the LIKE *button* option and you do not use the LABEL option, the button inherits the label of the button you name.

You can indicate a character within the label to be used as a navigation mnemonic in Windows. Indicate the character by preceding it with an ampersand (&). When the button is displayed, the mnemonic is underlined. The user can choose the button by pressing **ALT** and the underlined letter. If you specify more than one button with the same mnemonic, the AVM transfers focus to each of these in tab order when you make a selection.

To include a literal ampersand within a label, specify a double ampersand (&&).

LIKE *button*

Indicates the name of a defined button whose characteristics you want to use for a new button. If you name a button with this option, you must have defined that button earlier in the procedure. You can override the label, image, and on phrase by using the LABEL, IMAGE, and *on-phrase* options. If you do not use these options, the button takes on the characteristics of the button you name.

PFCOLOR *expression*

Specifies the prompt-for color for the button in character interfaces. This option is ignored in graphical interfaces.

size-phrase

Specifies the outside dimensions of the button widget. Following is the syntax for size-phrase:

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS }width BY height
```

If you specify SIZE or SIZE-CHARS, the units are characters; if you specify SIZE-PIXELS, the units are pixels. For character units, the values *width* and *height* must be decimal constants; for pixel units, they must be integer constants. For more information, see the SIZE phrase reference entry.

If no size is specified, the AVM calculates a default size for the button. This calculation adds the button's border thickness (that is, the combination of 3D shadows and highlights, and the focus rectangle) to the up image size defined by the IMAGE | IMAGE-UP *image-phrase* option. However, the thickness of the border depends on whether the button has dual images (up and down images) and whether it is a FLAT-BUTTON or NO-FOCUS button.

The following table explains how many pixels the image size expands based on the button size.

Table 28: Determining button border thickness

Button image	NO-FOCUS status	FLAT-BUTTON status	Border thickness
Up image only	No	No	7 pixels (2 pixels for the focus rectangle, 5 pixels for the 3D shading)
Up and down image	No	No	4 pixels (4 pixels for the focus rectangle, 0 pixels for the 3D shading)
Up image only	Yes	No	5 pixels (0 pixels for the focus rectangle, 5 pixels for the 3D shading)
Up and down image	Yes	No	0 pixels (ABL expects the specified image to include a border)
Up image only	Yes	Yes	2 pixels
Up and down image	Yes	Yes	2 pixels

`NO-FOCUS [ FLAT-BUTTON ]`

Specifies that the button should not accept focus. A button for which the NO-FOCUS attribute is defined will not take focus when the mouse is clicked on it, and it will not accept keyboard input. Also, the AVM will not generate ENTRY or LEAVE events for the button. NO-FOCUS buttons behave similarly to standard Windows toolbar buttons. The NO-FOCUS option is supported in Windows only.

A button with the NO-FOCUS attribute is not added to its parent frame's tab order. However, if the NO-FOCUS attribute is switched from TRUE to FALSE before the button is realized, the button is added to the end of its parent frame's tab order. Switching the NO-FOCUS attribute from FALSE to TRUE before realization removes the button from its parent frame's tab order.

Note: If a frame that contains a NO-FOCUS button does not itself have focus, the frame does not receive focus when the button is pushed. In this situation, frame entry or leave events are not generated. Focus stays on the current widget when a NO-FOCUS button is pushed, even across multiple frames in a window.

`FLAT-BUTTON`

A flat button is a new style of button which is two-dimensional until the mouse passes over it, at which time, a 3D border appears.

NO-CONVERT-3D-COLORS

Specifies that the colors of the button's images (that is, up, down, and insensitive) are not converted to the system 3D colors. By default, the AVM converts shades of gray in an image to the corresponding system 3D color. Using the NO-CONVERT-3D-COLORS option overrides this default behavior. The NO-CONVERT-3D-COLORS option is supported in Windows only.

The following table describes the conversion process.

Table 29: 3D-color conversions for buttons

If the color is . . .	And the original Red-Green-Blue (RGB) color value is . . .	Then the new converted system color is . . .
White	(255, 255, 255)	System button highlight color
Light Gray	(192, 192, 192)	System button face color
Dark Gray	(128, 128, 128)	System button shadow color
Black	(0, 0, 0)	System button text color

During a session, if Windows notifies the AVM that the system colors have changed, the button's images are re-loaded and converted to the new system colors, unless the NO-CONVERT-3D-COLORS option is specified.

TOOLTIP *tooltip*

Allows you to define a help text message for a button. The AVM automatically displays this text when the user pauses the mouse pointer over the button.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the ToolTip is removed from the button. No ToolTip is the default. ToolTips are supported in Windows only.

DROP-TARGET

Indicates whether you want to be able to drop a file onto the object.

trigger-phrase

Specifies application triggers for the button.

For more information, see the Trigger phrase reference entry.

Example

This procedure defines two buttons, positions the buttons within a form, assigns triggers to the buttons with ON statements, and enables the buttons by referencing them in an ENABLE statement:

r-button.p

```
DEFINE BUTTON more-button LABEL "More".
DEFINE BUTTON next-button LABEL "Next".

FORM more-button next-button
  WITH FRAME but-frame ROW 1.

FORM Customer.CustNum Customer.Name
  WITH FRAME brief ROW 4.

FORM Customer EXCEPT Customer.CustNum Customer.Name
  WITH FRAME full ROW 7.

ON CHOOSE OF more-button
  DISPLAY Customer EXCEPT Customer.CustNum Customer.Name WITH FRAME full.

ON CHOOSE OF next-button DO:
  HIDE FRAME full.
  FIND NEXT Customer NO-ERROR.
  DISPLAY Customer.CustNum Customer.Name WITH FRAME brief.
END.

FIND FIRST Customer.
DISPLAY Customer.CustNum Customer.Name WITH FRAME brief.

ENABLE more-button next-button WITH FRAME but-frame.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

When the procedure is run, the first Customer's number and name are initially displayed. The user can choose either the MORE button to see the entire Customer record or the NEXT button to see the next Customer's number and name.

The following example sets up a browse that allows you to drop a file on the browse:

```
DEFINE BUTTON button-1 LABEL ''Drop Here'' DROP-TARGET.
```

Notes

- When a frame receives a default RETURN event, it actually sends a CHOOSE event to the default button.
- To create the compile-time defined button you are defining, you must define a compile-time defined frame that contains the button. Each frame you define that contains the same button creates an additional instance of that button. The handle for a compile-time defined button is not available until the button is created.
- You must enable a button to make it available to the user. You can enable a button by setting its SENSITIVE attribute or by referencing it in an ENABLE or UPDATE statement.
- On a character-based terminal, a button appears as the label enclosed in angle brackets (< >). The user can move the mouse pointer to the button by pressing **TAB** or arrow keys. The user can then choose the button by pressing **SPACEBAR** or **RETURN**.
- You can specify an application-defined widget ID for a compile-time defined button widget using the *form-item* phrase in either the FORM statement or the DEFINE FRAME statement. See the FORM statement and DEFINE FRAME statement reference entries for more information.
- To make an application portable between graphical and character environments, you can specify an image and a label for a button. In graphical environments, the image is used and the label is ignored; in character environments, the label is used and the image is ignored.

- If you specify a size for a button, the button is not affected by changes to the size of any contained image. If you do not specify a size for the button, the button changes size to fit the image.
- In Windows, ABL supplies the following prepackaged images for the up, down, left, and right arrows: `btn-up-arrow`, `btn-down-arrow`, `btn-left-arrow`, and `btn-right-arrow`. Specify one of these items in place of a filename.

  Use these values for the IMAGE-UP option. Doing so makes the prepackaged image available to ABL in its up, down, and insensitive state, without specifying the IMAGE-DOWN and the IMAGE-INSENSITIVE options. You will also get appropriately sized arrows based on your screen resolution.
- You can apply entry to a NO-FOCUS button programmatically. The AVM does not report an error. However, the button will not respond to keyboard activity.
- The AVM draws the 3D effect only if a button has an up image, but no down image. If the button has both an up image and a down image, the AVM does not draw the 3D effect; the images, themselves, should be drawn with a 3D effect.
- The AVM only performs the color conversion process on bitmaps (`.bmp` files) that contain 256 or fewer colors. However, you might consider using 16-color bitmaps because only the first sixteen entries in the bitmap's color table will be converted.
- Icon colors (`.ico` files) are not converted, even if CONVERT-3D-COLORS is TRUE. To ensure that an icon will be displayed properly on a button, draw icons with a transparent background.

See also

Class-based data member access, FORM statement, Image phrase

DEFINE DATASET statement

Defines a ProDataSet object that is created at compile time, and allows you to identify the temp-table buffers it incorporates and define the data relations between those buffers. This statement defines the ProDataSet for use in one or more external procedures, or within a single class or class hierarchy.

Syntax

```
DEFINE { [ [ NEW ] SHARED ] | [ PRIVATE | PROTECTED ] [ STATIC ]
   [ SERIALIZABLE | NON-SERIALIZABLE ] } DATASET dataset-name
  [ NAMESPACE-URI namespace ] [ NAMESPACE-PREFIX prefix ]
  [ XML-NODE-NAME node-name ] [ SERIALIZE-NAME serialize-name ]
  [ XML-NODE-TYPE node-type ] [ SERIALIZE-HIDDEN ]
  [ REFERENCE-ONLY ] FOR buffer-name [ , buffer-name ] . . .
  [ DATA-RELATION [data-rel-name ] FOR data-rel-spec ] . . .
  [ PARENT-ID-RELATION [data-rel-name] FOR parent-id-rel-spec ] . . .
```

NEW SHARED DATASET *dataset-name*

Defines and identifies a ProDataSet object that can be shared by one or more procedures called directly or indirectly by the current procedure. The ProDataSet object remains available to other procedures until the procedure that defined it ends. The called procedures must define the same ProDataSet object name as SHARED. For shared ProDataSet objects, each *buffer-name* must be the name of a shared buffer.

Note: Because a static SHARED TEMP-TABLE cannot have a BEFORE-TABLE, you cannot track changes on a static SHARED DATASET.

SHARED DATASET *dataset-name*

Defines and identifies a ProDataSet object that was initially defined by another procedure as NEW SHARED. For shared ProDataSet objects, each *buffer-name* must be the name of a shared buffer.

The procedure that initially defines the object determines the name. The procedures that share the object must define the object with the same name.

[PRIVATE **|** PROTECTED **][** STATIC **][** SERIALIZABLE **|** NON-SERIALIZABLE **]**
DATASET *dataset-name*

Defines and identifies a ProDataSet object as a data member of a class, and optionally specifies an access mode (PRIVATE or PROTECTED) and scope (instance or STATIC) for that data member, as well as whether or not the ProDataSet object will participate in serialization (SERIALIZABLE or NON-SERIALIZABLE). You cannot specify any of these options for a ProDataSet defined in an interface type definition (INTERFACE statement block) or when defining a ProDataSet as a data element of a procedure. For a data member ProDataSet, each *buffer-name* must be the name of a compatible data member buffer (see the FOR option).

Note: The specified options are applicable only when defining a data member for a class in a class definition (`.cls`) file. The PRIVATE/PROTECTED modifier and the STATIC modifier can appear in either order, e.g., `DEFINE STATIC PRIVATE DATASET myDataSet...` is valid syntax. You cannot shadow (override) the definition of a given ProDataSet data member in a class hierarchy.

A ProDataSet defined with the STATIC option is a static data member of the class type for which it is defined, and it is scoped to the ABL session where it is referenced. ABL creates one copy of the specified class static ProDataSet on first reference to the class type, and ABL creates only one such copy for any number of instances of the class that you create. You cannot specify STATIC if you specify the REFERENCE-ONLY option. You can directly reference an accessible static ProDataSet data member from any other static or instance class member defined in the same class or class hierarchy.

Without the STATIC option, ABL creates an instance ProDataSet data member that is scoped to a single instance of the class where it is defined. ABL creates one copy of the specified instance ProDataSet for each such class instance that you create. You cannot directly reference an instance ProDataSet data member from a STATIC class member definition defined within the same class or class hierarchy.

For more information on accessing ProDataSets of different access modes and scopes, see the reference entry for Class-based data member access.

Class-based objects that are defined as serializable (using the SERIALIZABLE option in the CLASS statement) can be passed as parameters in remote calls between the application server and ABL clients and can be serialized to binary or JSON format. By default, serializing an object to binary via the `Progress.IO.BinarySerializer` class includes all data members regardless of access mode. However, for JSON serialization via `Progress.IO.JsonSerializer`, only public data members are serialized. To include a protected or private ProDataSet object during JSON serialization, SERIALIZABLE must be added to the definition. See Serialize( ) method (JsonSerializer) for more information.

Use the NON-SERIALIZABLE option to exclude a given ProDataSet from parameter passing between an application server and ABL client and the serialization process via the `Progress.IO.BinarySerializer` class. (ProDataSets are NON-SERIALIZABLE by default via the `Progress.IO.JsonSerializer` class because they cannot be defined as public.)

Note: Members of a class are grouped into six namespaces, including buffers/temp-tables, methods, variables/properties/events, ProDataSets, queries, and data-sources. ProDataSets defined as members of a class share the same namespace. There can be only one class member in this namespace with a given name.

For more information on where and how to define data members in a class, see the CLASS statement reference entry.

`DATASET` *`dataset-name`*

Defines and identifies a ProDataSet object in an interface, or for access only within the current external procedure or as a data member of the current class.

`NAMESPACE-URI` *`namespace`*

An optional CHARACTER constant that specifies the URI for the namespace of the ProDataSet object.

`NAMESPACE-PREFIX` *`prefix`*

An optional CHARACTER constant that specifies the namespace prefix associated with the NAMESPACE-URI.

`XML-NODE-NAME` *`node-name`*

An optional CHARACTER constant that specifies the name of the XML element representing the ProDataSet in an XML Document. The default is *`dataset-name`*. This option is useful when:

- reading a ProDataSet from XML with a name that contains invalid characters for an ABL name or is an ABL keyword, or
- writing a ProDataSet to XML with a name that is an invalid XML tag name.

Note: If you set SERIALIZE-NAME but do not set XML-NODE-NAME, the AVM sets XML-NODE-NAME equal to SERIALIZE-NAME.

`XML-NODE-TYPE` *`node-type`*

An optional CHARACTER constant that specifies the XML node type of the ProDataSet element represented in XML. The valid option values are `"ELEMENT"` and `"HIDDEN"`. The default value is `"ELEMENT"`.

SERIALIZE-HIDDEN

Indicates that the ProDataSet name is not written when the ProDataSet data or schema is serialized to JSON or XML. (This option is ignored for serialization via Progress.IO.JsonSerializer.)

This will support the WRITE-JSON( )and WRITE-XML( ) methods in case the root of the JSON string or XML document maps to a temp-table. For XML, the XML-NODE-TYPE attribute takes precedence over SERIALIZE-HIDDEN attribute.

SERIALIZE-NAME *serialize-name*

An optional CHARACTER constant that specifies the name of the element representing the ProDataSet in an XML Document or JSON string. The default is *dataset-name*. This option is useful when:

- reading a ProDataSet from XML or JSON with a name that contains invalid characters for an ABL name or is an ABL keyword, or
- writing a ProDataSet to XML or JSON with a name that is an invalid XML tag name or an invalid JSON property name.

Note: If you set both XML-NODE-NAME and SERIALIZE-NAME, the READ-XML() and WRITE-XML() methods will use XML-NODE-NAME.

REFERENCE-ONLY

Specifies that the procedure or class defining this ProDataSet object is using the object definition only as a reference to a ProDataSet object that is defined and instantiated in another procedure or class, and specified as a parameter in the invocation of a RUN statement, a method in a class, or a user-defined function, using either the BY-REFERENCE or BIND option. The AVM does not instantiate the reference-only object. You cannot specify REFERENCE-ONLY if you specify the STATIC option.

Passing a reference-only ProDataSet object parameter to a local routine, using either the BY-REFERENCE or BIND option allows the calling routine and the called routine to access the same object instance (instead of deep-copying the parameter).

Note: If you pass the parameter to a remote procedure, the AVM deep-copies the parameter on OUTPUT and the reference-only parameter is bound to that copy.

When you pass a ProDataSet parameter to a local routine using the BY-REFERENCE option, both the calling and called routines access the calling routine's object instance (and ignore the called routine's object instance). Since the called routine's object instance is ignored, you should define the object as a reference-only object. When you define a reference-only ProDataSet object in the called routine and receive it from the calling routine using the BY-REFERENCE option, the AVM binds the definition of the object in the called routine to the object instance in the calling routine for the duration of the called routine. You cannot define a reference-only ProDataSet object in the calling routine and pass it to the called routine using the BY-REFERENCE option.

When you pass a ProDataSet parameter to a local routine using the BIND option, you can define a reference-only ProDataSet object in either the calling routine or the called routine as follows:

- When you define a reference-only ProDataSet object in the calling routine and pass it to the called routine using the BIND option, the AVM binds the calling routine to the object instance in the called routine. The reference-only object definition remains bound to the object instance until the routine containing the reference-only object definition is deleted or terminates. The parameter must be an OUTPUT parameter.

Note: If you also define the ProDataSet object instance in the called routine as a reference-only object, you must bind the object in the called routine before returning to the calling routine.

- When you define a reference-only ProDataSet object in the called routine and receive it from the calling routine using the BIND option, the AVM binds the called routine to the object instance in the calling routine. The reference-only object definition remains bound to the object instance until the routine containing the reference-only object definition is deleted or terminates. The parameter must be an INPUT or INPUT-OUTPUT parameter.

In either case, you must specify the BIND option for the parameter in both the invocation of a RUN statement, a method in a class, or a user-defined function, and in the DEFINE PARAMETER statement.

Caution: Do not delete the object or routine to which a reference-only ProDataSet object is bound, or you might be left with references to an object that no longer exists.

When you define a ProDataSet object as reference-only, you must also define all member temp-tables as such. A temp-table object defined as reference-only can be a member of a reference-only ProDataSet object or a standard ProDataSet object. However, if you define a reference-only temp-table in a standard ProDataSet object, you cannot use the ProDataSet object until you bind the reference-only temp-table.

FOR *buffer-name* **[** , *buffer-name* **]...**

Specifies the name of compile-time defined temp-table buffer that is scoped to the procedure or class in which the ProDataSet object is defined. For a shared ProDataSet, each *buffer-name* must be a shared buffer. If the ProDataSet is a data member of a class, you must specify the name of a compatible buffer data member. Thus, if the ProDataSet is a static data member, the buffer must also be a static data member; if the ProDataSet is PROTECTED, the buffer must also be defined as PROTECTED or inherited from a super class; and if the ProDataSet is PRIVATE, the buffer can be defined as either PRIVATE or PROTECTED. If the ProDataSet is a PRIVATE instance data member, you can also specify a default database table buffer; you cannot specify a default database buffer for a ProDataSet data member defined with any other combination of access mode and scope.

Note: ABL defines the default buffer for every database table that you access in a class as a PRIVATE instance buffer data member.

DATA-RELATION **[***data-rel-name* **]** FOR *data-rel-spec*

Specifies an optional data-relation object.

The *data-rel-name* argument lets you name the data-relation object. You can use this name to obtain the object's handle at run time. This argument is optional. The default name is Relation*n* (where *n* starts at 1 for each ProDataSet object).

The*data-rel-spec* argument specifies a pair of parent and child buffers for the data-relation object using the following syntax:

```
parent-buffer-name, child-buffer-name field-mapping-phrase
[ REPOSITION ][ NESTED [ FOREIGN-KEY-HIDDEN ]][ NOT-ACTIVE ]
[ RECURSIVE ]
```

parent-buffer-name

The parent buffer in the data-relation object. This must be one of the buffers identified by *buffer-name*.

child-buffer-name

The child buffer in the data-relation object. This must be one of the buffers identified by *buffer-name*.

field-mapping-phrase

Specifies the fields in the relation using the following syntax:

```
RELATION-FIELDS ( parent-field1 , child-field1
  [ , parent-fieldn , child-fieldn ] . . . )
```

The first field in the pair is from the parent buffer, the second field is from the child buffer. When filling the ProDataSet object, the AVM retrieves data for the child buffer based on an equality match between all pairs of fields unless the Data-Relation is deactivated or there is an explicit query definition for the data source of the child buffer.

You can define a query for the data source of the child buffer, or supply custom logic in response to FILL events that take over complete responsibility for filling one level of the ProDataSet object. In these cases, the *field-mapping-phrase* is not used.

When navigating the ProDataSet object, the AVM filters the data in the child buffer to include only children of the current parent.

REPOSITION

Specifies the relation mode as REPOSITION. The relation mode is SELECTION, by default.

When the relation mode is SELECTION, the ProDataSet object fills the child temp-table of the relation with all records related to the current parent. When the relation mode is REPOSITION, the relation is effectively ignored during a FILL, and the child of the relation is treated as if it were a top-level buffer.

When navigating a filled ProDataSet object with a SELECTION relation, related data is filtered as it is browsed. This means the child query of the relation is filtered to make available only children of the current parent, and the query is re-opened each time the parent table is repositioned. When navigating a filled ProDataSet object with a REPOSITION relation, the child table query is always set to match all the rows in the child table, and is not re-opened when the parent changes. Only the buffer for the child is repositioned to the matching child for the current parent.

A data-relation defined with REPOSITION or SELECTION is overridden if a query is defined for a child relation data-source. Normally, if REPOSITION is not specified, the child query selects the children of the parent. Coding a query for the child data-source overrides whether or not a SELECTION or REPOSITION relation mode was defined. By removing the query of the child data-source, the default child table load occurs.

NESTED

Specifies that child rows of a ProDataSet buffer are nested within their parent rows when writing a JSON representation of data or an XML representation of data or schema. This also causes the XML Schema definitions for the related temp-tables to be nested.

Note: If you do not nest child tables, the JSON string loses the data relation information. The serialization process also loses any information about key columns.

FOREIGN-KEY-HIDDEN

Specifies that foreign key fields in nested data relations are omitted when writing JSON or XML data, and that foreign key fields are optional in the XML Schema representation of the child table definition. This option can only be specified if the NESTED option is also specified.

When working with large ProDataSets, omitting foreign keys in nested child records can yield smaller JSON strings and XML documents, more efficient network transfers, and performance gains with the READ-*( ) and WRITE-*( ) methods.

Think carefully before deciding to use this feature. The READ-*( ) methods automatically populate foreign keys in nested child records with the value in the outer parent record when the foreign key is omitted from the serialized data. Unless you are sure that the data's receiver will do the same, you should not specify FOREIGN-KEY-HIDDEN in your nested data-relations.

For example, while .NET can read this XML document and populate an ADO .NET Dataset, it will create rows in the child DataTable with a null value for the foreign key field. This is because it does not recognize the data-relation definition in the XML Schema as NESTED. ABL uses the prodata:nested attribute in the XML Schema. .NET only recognizes its own msdata:IsNested attribute.

NOT-ACTIVE

Specifies that the data-relation is inactive. For example, this allows you to have two relations between the same two ProDataSet temp-table buffers, but only have one relation active at a time.

RECURSIVE

Instructs the ProDataSet FILL to load self-referencing elements, such as bill-of-materials parent/child tables or organization chart manager/employee tables. Support is provided for a recursive data-relation during a FILL for a ProDataSet or temp-table buffer. Navigation is not supported, but can be done normally through .NET, Java, or a recursive ABL procedure.

PARENT-ID-RELATION [*data-rel-name*] FOR *parent-id-rel-spec*

Specifies that:

- The relationship between the parent and child buffer is based on the RECID of the parent record when reading from or writing to an XML document or JSON string.
- The child rows of a ProDataSet buffer are nested within their parent rows when the ProDataSet data or schema is rendered to an XML document or JSON string.
- The child RECID field is not serialized.

data-rel-name

An optional argument that lets you name the data-relation. You can use this name to obtain the PARENT-ID-RELATION's handle at run time. The default value is Relation*n* (where *n* starts at 1 for each ProDataSet).

parent-id-rel-spec

Specifies a pair of parent and child buffers for the PARENT-ID-RELATION, the RECID field in the child buffer, and the XML and JSON serialization order for parent record fields using the following syntax:

```
parent-buffer-name, child-buffer-name PARENT-ID-FIELD id-field
[ PARENT-FIELDS-BEFORE ( parent-field1
  [ , parent-field2 ] . . . ) ]
[ PARENT-FIELDS-AFTER ( parent-field1
  [ , parent-field2 ] . . . ) ]
```

parent-buffer-name

Specifies the name of the parent buffer in PARENT-ID-RELATION. This is one of the buffers identified by *buffer-name*.

child-buffer-name

Specifies the child buffer in PARENT-ID-RELATION. This is one of the buffers identified by *buffer-name*.

PARENT-ID-FIELD *id-field*

Specifies the RECID field in *child-buffer-name*. When writing records from the *parent-buffer-name* temp-table to XML or JSON, *child-buffer-name* records whose id-field's value is equal to the RECID of the parent record will be nested within the parent record's XML node or JSON string.

PARENT-FIELDS-BEFORE (*parent-field1* [, *parent-field2*] . . .)

An optional phrase that specifies the fields from the *parent-buffer-name* temp-table record that are to be written to XML before any *child-buffer-name* temp-table records.

PARENT-FIELDS-AFTER (*parent-field1* [, *parent-field2*] . . .)

An optional phrase that specifies the fields from the *parent-buffer-name* temp-table record that are to be written to XML after all *child-buffer-name* temp-table records.

When neither PARENT-FIELDS-BEFORE or PARENT-FIELDS-AFTER are specified, the default behavior is to write all the fields from the parent record to XML before any nested child records.

If a PARENT-ID-RELATION contains PARENT-FIELDS-BEFORE or PARENT-FIELDS-AFTER, then all the fields from the parent buffer that have XML-NODE-TYPE as ELEMENT must be listed among the PARENT-ID-RELATIONs containing the buffer as *parent-buffer-name*. Else, the AVM raises a compiler error. The compiler also checks for duplicate entries.

Notes

- PARENT-FIELDS-BEFORE is only allowed on the first PARENT-ID-RELATION for *parent-buffer-name*. If PARENT-FIELDS-BEFORE appears in a subsequent PARENT-ID-RELATION for *parent-buffer-name,* the AVM raises a compiler error.

- A ProDataSet defined with the `PARENT-ID-RELATION` option cannot be passed as a parameter to a procedure or user-defined function that defines an OpenEdge Web service. If you do, ProxyGen fails.

Example

For examples of ProDataSet definitions and usage, see *Use ProDataSets*.

For examples of instance and static ProDataSet data member definitions, see the descriptions of `r-CustObj.cls`, `r-CustObjStatic.cls`, and `r-CustObjAbstract.cls` in the CLASS statement reference entry.

Notes

- You cannot define a SHARED or NEW SHARED ProDataSet object in a class definition (`.cls`) file. If you do, ABL generates a compilation error.
- You cannot define a ProDataSet object within an internal procedure, a user-defined function, or a method of a class (including constructors, destructors, and property accessors).
- A ProDataSet can be compile-time defined (often referred to as a *static* ProDataSet object), where the ProDataSet is defined and created at compile time using this statement; or it can be run-time defined (often referred to as a *dynamic* ProDataSet object), where the ProDataSet is defined and created at run time using the CREATE DATASET statement and ProDataSet object handle operations. A compile-time defined ProDataSet can also be defined as a static data member of a class. In this case, it is a static ProDataSet object that is also a class static data member.
- A ProDataSet can have no more than one recursive data-relation.

See also

Buffer object handle, Class-based data member access, CREATE DATASET statement, Data-relation object handle, DEFINE QUERY statement, DEFINE TEMP-TABLE statement, GET-TOP-BUFFER( ) method, NUM-REFERENCES attribute, OPEN QUERY statement, ProDataSet object handle, RECURSIVE attribute

DEFINE DATA-SOURCE statement

Defines a data-source object that is created at compile time for use in an external procedure or within a single class or class hierarchy.

Syntax

```
DEFINE [ PRIVATE | PROTECTED ] [ STATIC ] DATA-SOURCE data-source-name
  FOR [ QUERY query-name ]
      [ source-buffer-phrase [ , source-buffer-phrase ] . . . ]
```

[PRIVATE | PROTECTED] [STATIC] DATA-SOURCE *data-source-name* FOR

Defines and identifies a data-source object as a data member of a class, and optionally specifies an access mode (PRIVATE or PROTECTED) and scope (instance or STATIC) for that data member. You cannot specify these options when defining a data-source as a data element of a method (including constructors, destructors, and property accessors) or a procedure.

Note: The specified options are applicable only when defining a data member for a class in a class definition (`.cls`) file. The PRIVATE/PROTECTED modifier and the STATIC modifier can appear in either order, e.g., `DEFINE STATIC PRIVATE DATA-SOURCE myDataSource...` is valid syntax. Note also that you cannot shadow (override) the definition of a given data-source data member in a class hierarchy.

PRIVATE data-source data members can be accessed by the defining class. An instance can access a private data-source data member of another instance if both instances are of the same class. PROTECTED data-source data members can be accessed by the defining class and any of its derived classes. An instance can access the protected data-source data member of a second instance of a class that is at the same level or higher in the class hierarchy. The default access mode is PRIVATE.

A data-source defined with the STATIC option is a static data member of the class type for which it is defined and is scoped to the ABL session where it is referenced. ABL creates one copy of the specified class static data-source on first reference to the class type, and ABL creates only one such copy for any number of instances of the class that you create. You can directly reference an accessible static data-source data member from any other static or instance class member defined in the same class or class hierarchy.

Without the STATIC option, ABL creates an instance data-source data member that is scoped to a single instance of the class where it is defined. ABL creates one copy of the specified instance data-source for each such class instance that you create. You cannot directly reference an instance data-source data member from a STATIC class member definition defined within the same class or class hierarchy.

For more information on accessing data-sources of different access modes and scopes, see the reference entry for Class-based data member access.

Note: Members of a class are grouped into six namespaces, including buffers/temp-tables, methods, variables/properties/events, ProDataSets, queries, and data-sources. Data-sources defined as members of a class share the same namespace. There can be only one class member in this namespace with a given name.

For more information on where and how to define data members in a class, see the CLASS statement reference entry.

`DATA-SOURCE` *`data-source-name`* `FOR`

Defines and identifies a data-source object for access only within the current procedure, method of a class (including constructors, destructors, and property accessors), or as a PRIVATE data member of a class.

[`QUERY` *`query-name`* **]**

Specifies a query that defines the buffers for the data-source object. Use this option to override the default query for the data-source object.

You can also use the QUERY attribute to override the default query, or the FILL-WHERE-STRING attribute to override the WHERE clause in the query.

If the data-source is a data member of a class, you must specify the name of a compatible query. Thus, if the data-source is a static data member, the query must also be a static data member; if the data-source is PROTECTED, the query must also be defined as PROTECTED or inherited from a super class; and if the data-source is PRIVATE, the query can be defined as either PRIVATE or PROTECTED.

source-buffer-phrase [, *source-buffer-phrase*] . . .

Specifies one or more database or temp-table buffers for the data-source object using the following syntax:

buffer-name [KEYS ({ *field1* [, *fieldn*] . . . | ROWID })]

buffer-name

The name of the database or temp-table buffer. If the data-source is a data member of a class, you must define and specify the name of a compatible buffer data member for the table. Thus, if the data-source is a static data member, the buffer must also be a static data member; if the data-source is PROTECTED, the buffer must also be defined as PROTECTED or inherited from a super class; and if the data-source is PRIVATE, the buffer can be defined as either PRIVATE or PROTECTED. If the data-source is a PRIVATE instance data member, you can also specify a default database table buffer; you cannot specify a default database buffer for a data-source data member defined with any other combination of access mode and scope.

Note: ABL defines the default buffer for every database table that you access in a class as a PRIVATE instance buffer data member.

KEYS ({ *field1* [, *fieldn*] . . . | ROWID })

Specifies one or more database table fields that constitute a unique key that can be used by the AVM to find a record in the table given a record in the temp-table that uses it as a data source.

The ROWID keyword can occur exactly once in place of a field list to use the ROWID as the key.

Examples

For examples of data-source definitions and usage, see *Use ProDataSets*.

For examples of instance and static data-source data member definitions, see the descriptions of `r-CustObj.cls`, `r-CustObjStatic.cls`, and `r-CustObjAbstract.cls` in the CLASS statement reference entry.

Note

A data-source can be compile-time defined (often referred to as a *static* data-source object), where the data-source is defined and created at compile time using this statement, or it can be run-time defined (often referred to as a *dynamic* data-source object), where the data-source is defined and created at run time using the CREATE DATA-SOURCE statement and data-source object handle operations. A compile-time defined data-source can also be defined as a static data member of a class. In this case, it is a static data-source object that is also a class static data member.

See also

Class-based data member access, CREATE DATA-SOURCE statement, Data-source object handle, FILL( ) method, FILL-WHERE-STRING attribute, QUERY attribute

DEFINE ENUM statement

See ENUM statement.

DEFINE EVENT statement

Defines an ABL class event, declares a class event prototype in an ABL interface, or overrides an abstract class event inherited from an ABL or .NET abstract super class. A class event is a member of the class in which it is defined. You can publish a non-abstract event from within the defining class definition using the built-in Publish( ) event method in response to a condition that you determine. You can subscribe a class method or internal procedure as a handler for any accessible event (abstract or non-abstract) using the built-in Subscribe( ) event method. This handler executes whenever the event is published. You can also unsubscribe any handler using the built-in Unsubscribe( ) event method. The following description begins with general syntax for defining a class event.

Note: This statement is applicable only when used in a class or interface definition (`.cls`) file. For more information on class events, see the Class Events Reference.

Syntax

```
DEFINE [ PRIVATE | PACKAGE-PRIVATE | PROTECTED | PACKAGE-PROTECTED | PUBLIC ]
  [ STATIC | ABSTRACT ][ OVERRIDE ]
  EVENT event-name signature-spec
```

Use the following syntax to declare a class event prototype in an interface:

```
DEFINE [ PUBLIC ] EVENT event-name signature-spec
```

Use the following syntax to declare an abstract class event prototype:

```
DEFINE [ PACKAGE-PRIVATE | PROTECTED | PACKAGE-PROTECTED | PUBLIC ]
  [ OVERRIDE ] ABSTRACT
  EVENT event-name signature-spec
```

`[ PRIVATE | PACKAGE-PRIVATE | PROTECTED | PACKAGE-PROTECTED | PUBLIC ]`

Specifies the access mode for this event. For a class event, the access mode indicates what code can call the Subscribe( ) event method on the event.

Note: You can only directly publish an event from within the class where it is defined, regardless of its access mode.

A PRIVATE event can have a handler subscribed to it only by the defining class. An instance can subscribe a handler to a private event of another instance if both instances are of the same class. A PACKAGE-PRIVATE event can have a handler subscribed to it by the defining class and any class within its package. A PROTECTED event can have a handler subscribed to it by the defining class and any of its derived classes. A PACKAGE-PROTECTED event can have a handler subscribed to it by the defining class, any class within its package, and from within any subclass that inherits the class. A PUBLIC event can have a handler subscribed to it from any code (in a class or a procedure) that has access to a class instance.

The default access mode is PUBLIC.

When declaring an interface event prototype, the access mode for the event must be PUBLIC (the default).

When defining an abstract event, the access mode for the event cannot be PRIVATE.

Note: The access mode modifier and the STATIC/ABSTRACT modifier can appear in either order, e.g., `DEFINE STATIC PUBLIC event1 EVENT...` is valid syntax.

[`STATIC`]

Defines an event that is a static member of the class type for which it is defined and that is scoped to the ABL session where it is referenced. ABL creates one copy of the specified class static event on first reference to the class type, and ABL creates only one such copy for any number of instances of the class that you create. You can subscribe a handler for an accessible static event in any piece of code. You can publish a static event only in an instance or static method that is defined within the same class definition where the static event is defined.

Without this option, ABL defines an instance event that is scoped to a single instance of the class where it is defined. ABL creates one copy of the specified instance event for each such class instance that you create. You can subscribe a handler for any public instance event (abstract or non-abstract) in any procedure, or in any instance or static method defined inside or outside of the class where the instance event is defined. Any static method can publish the public instance event only using an object reference to a class instance that defines the event as a member. If the referencing static method is defined in the same class as the public instance event, the class must instantiate itself in order to have access to an instance reference.

You can publish a non-abstract instance event only in a method that is defined within the same class definition where the instance event is defined. If the method is static, the instance event must also be public and you can only publish it using an object reference to an instance of the class. An instance method defined in the same class can also publish a public instance event using an object reference to an instance of the class.

For more information on the mechanism for subscribing handlers to static and instance class events, see the Subscribe( ) event method reference entry. Note that the same constraints on subscribing handlers for static and instance class events applies to unsubscribing the handlers. For more information, see the Unsubscribe( ) event method. For more information on the mechanism for publishing instance and static class events, see the Publish( ) event method reference entry.

The STATIC option is not valid when you:

- Define or implement an abstract event
- Declare an interface event prototype
- Implement an interface event

[ABSTRACT **]**

Defines the event as an abstract instance member of the class type for which it is defined. The defining class type must also be abstract. If you define an abstract event, it has the following requirements:

- You must specify the OVERRIDE option if an inherited abstract event has the same name and signature.

  Note: Class events cannot be overloaded. If you want to define an event with a different signature, it must also have a different name than any other event inherited by or defined in the same class.

- You can specify any access mode other than PRIVATE.
- The abstract event must be overridden and implemented in a derived class.

Any class definition for an instance event that includes the OVERRIDE option and does not include the ABSTRACT option defines an event implementation. If it also includes the ABSTRACT option, the inherited abstract event remains abstract.

Note: You cannot publish an event that is defined as ABSTRACT.

This option is not valid either when you define a static event or when you declare an interface event prototype.

[OVERRIDE **]**

Specifies that this instance event overrides an abstract event inherited from an ABL or .NET class.

Note: A .NET abstract event or class is defined in C# with the `abstract` keyword.

When you specify OVERRIDE, *`event-name`* must be identical to the name of the overridden abstract event, and *`signature-spec`* must specify a signature that is identical to the signature defined for the overridden event. In addition, the access mode must not be more restrictive than the access mode defined for the overridden event. When overriding a .NET event, the *`signature-spec`* must specify the same delegate that is defined for the .NET event. For more information, see the description of the *`signature-spec`* option.

If you specify the ABSTRACT option, your overriding event is also defined as abstract, and it must be implemented in a class derived from the defining class. Note that you do not have to override an inherited abstract event that you want to remain abstract as long as the inheriting class is also abstract. However, doing so allows you to specify a less restrictive access mode for the abstract event.

If you do not specify the ABSTRACT option, your overriding event implements the inherited abstract event.

This option is not valid:

- When you define a static event
- When you declare an interface event prototype
- To override an inherited event that is implemented (not abstract)

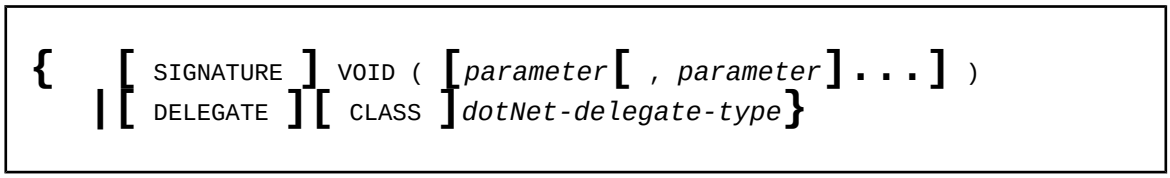
EVENT *event-name*

Specifies the event name. This event name must be unique among all events, properties, and variable data members defined in the class hierarchy of the defining class, unless you specify the OVERRIDE option to override an inherited abstract event.

Note: Members of a class are grouped into six namespaces, including buffers/temp-tables, methods, variables/properties/events, ProDataSets, queries, and data-sources. Variables, properties, and events defined as members of a class share the same namespace. There can be only one class member in this namespace with a given name (not counting abstract member overrides).

signature-spec

Specifies the signature for the Publish( ) event method and for any class method or internal procedure that executes as a handler when the event is published. You can define this signature using one of the two options in the following syntax:



[SIGNATURE] VOID ([*parameter*[, *parameter*]...])

Defines an ABL method signature for the event, including the return value and zero or more parameters. The return value is always VOID. No associated internal procedure or method event handler can return a value. The SIGNATURE keyword is optional for readability.

If this event implements an ABL interface event or overrides an inherited ABL abstract event (using the OVERRIDE option), the interface or inherited abstract event must also be defined with an ABL method signature that matches this signature. If the ABL interface or inherited ABL abstract event is defined with a .NET signature (*dotNet-delegate-type*), you must use the .NET signature option to define this event instead.

For more information on the syntax of *parameter* and on matching parameters to implement or override ABL class events, see the Parameter definition syntax reference entry.

[DELEGATE][CLASS]*dotNet-delegate-type*

Defines a .NET signature for the event as specified by a .NET delegate type (*dotNet-delegate-type*), for example, `System.EventHandler`. You can specify the delegate class name without a namespace (for example, `EventHandler`) with the presence of an appropriate USING statement. The DELEGATE or CLASS keyword is optional for readability. However the CLASS keyword, only, also disambiguates a delegate type name that might be identical to an ABL primitive type name. Note that ABL only supports .NET delegate types that conform to the .NET Framework convention for event handlers. This convention defines an event handler signature with a VOID return type and two input parameters, where the first parameter is a `System.Object` and the second parameter is a `System.EventArgs` or a .NET class derived from `System.EventArgs`. The .NET delegate type can also be a constructed generic type. For more information on .NET generic types, see the Data types reference entry.

You must use this option to define the ABL class event signature if the ABL event implements a .NET interface event or overrides an inherited .NET abstract event (using the OVERRIDE option). Also, the *dotNet-delegate-type* for the ABL event must be identical to the .NET delegate type used to define the .NET interface or abstract event.

You can use this option to define the signature for an ABL class event even if the event does not implement or override a .NET event. However, if this event implements an ABL interface event or overrides (using the OVERRIDE option) an inherited ABL abstract event that is, itself, defined with a .NET delegate, you must also define the signature for this event using an identical *dotNet-delegate-type*.

Note that you can identify the data types of .NET event parameters by looking up the delegate type defined for the event in the appropriate .NET event documentation.

Examples

The following sample class and procedure files define, publish, subscribe an event handler to, and unsubscribe an event handler from an ABL class event. The `r-EventPublish` sample class defines the public `NewCustomer` event and a `PubNewCustomer( )` method to publish it.

r-EventPublish.cls

```
CLASS r-EventPublish:

  /* Define an event */
  DEFINE PUBLIC EVENT NewCustomer
    SIGNATURE VOID ( INPUT pcCustName AS CHARACTER ).

  /* Code that publishes the event. */
  METHOD PUBLIC VOID PubNewCustomer( ):

    DEFINE VARIABLE cCustName AS CHARACTER INITIAL "A Customer Name" NO-UNDO.

    NewCustomer:Publish( INPUT cCustName ) NO-ERROR.

  END METHOD.

END CLASS.
```

The `r-EventSubscribe` sample class defines and subscribes a `NewCustomer( )` method as a handler for the event when it is instantiated, based on the object reference to the `r-EventPublish` class that is passed to the constructor. Note that the event handler also unsubscribes itself to the event after it executes.

r-EventSubscribe.cls

```
CLASS r-EventSubscribe:

  DEFINE VARIABLE rPubObj AS CLASS r-EventPublish NO-UNDO.

  CONSTRUCTOR PUBLIC r-EventSubscribe
    ( INPUT prPubObj AS CLASS r-EventPublish):

    ASSIGN rPubObj = prPubObj.
    rPubObj:NewCustomer:Subscribe( NewCustomer_Handler ) NO-ERROR.

  END CONSTRUCTOR.

  /* Method used as event handler */
  METHOD PUBLIC VOID NewCustomer_Handler ( INPUT pcCustName AS CHARACTER ):

    MESSAGE "Subscriber received event NewCustomer" SKIP
            "CustName =" pcCustName VIEW-AS ALERT-BOX.

    rPubObj:NewCustomer:Unsubscribe( NewCustomer_Handler ) NO-ERROR.

  END METHOD.

END CLASS.
```

The `r-EventPubSub.p` sample procedure instantiates these classes. The procedure then displays a frame that includes a `bNewCust` button that when clicked runs an internal procedure, which in turn invokes the `PubNewCustomer( )` method on the `r-EventPublish` object to publish the event. This demonstrates how the class defining an event can allow a client class or procedure to publish it by providing a public method for the purpose. Note that any subsequent attempt to click the button does not run the event handler, because the handler has unsubscribed itself.

r-EventPubSub.p

```
DEFINE VARIABLE rPubObj AS CLASS r-EventPublish NO-UNDO.
DEFINE VARIABLE rSubObj AS CLASS r-EventSubScribe NO-UNDO.

DEFINE BUTTONbNewCust LABEL "New Customer".
DEFINE BUTTON bQuit    LABEL "Quit".

FORM bNewCust bQuit WITH FRAME aFrame.

ON CHOOSE OF bNewCust RUN CallNewCust NO-ERROR.

rPubObj = NEW r-EventPublish( ).
rSubObj = NEW r-EventSubScribe( rPubObj ).

ENABLE ALL WITH FRAME aFrame.
WAIT-FOR CHOOSE OF bQuit OR WINDOW-CLOSE OF CURRENT-WINDOW.

PROCEDURE CallNewCust:

  /* Call to publish */
  rPubObj:PubNewCustomer( ) NO-ERROR.

END PROCEDURE.
```

For more examples of class event definitions, including static and abstract events, see the descriptions of `r-CustObj.cls`, `r-CustObjStatic.cls`, and `r-CustObjAbstract.cls` in the CLASS statement reference entry.

Notes

- For information on using class event handlers and the built-in event methods, including Publish( ), Subscribe( ), and Unsubscribe( ), and on .NET see the Class Events Reference.
- If you define an ABL event using a .NET delegate, .NET can only subscribe to this event if its definition implements or overrides an event defined in a .NET interface or abstract class.

See also

CLASS statement, INTERFACE statement, Parameter definition syntax, Publish( ) event method, Subscribe( ) event method, Type-name syntax, Unsubscribe( ) event method, USING statement

DEFINE FRAME statement

Defines and creates a frame or dialog box that is created at compile time for use in one or more procedures, or within a single class.

Syntax

```
DEFINE {[[ NEW ] SHARED ]|[ PRIVATE ]} FRAME frame
  [form-item...]
  [{ HEADER | BACKGROUND }head-item...]
  {[frame-phrase]}
```

```
DEFINE {[[ NEW ] SHARED ]|[ PRIVATE ]} FRAME frame
  record[ EXCEPT field...]
  {[frame-phrase]}
```

`NEW SHARED FRAME` *frame*

Defines and identifies a frame to be shared by a procedure called directly or indirectly by the current procedure. The called procedure must name the same frame in a DEFINE SHARED FRAME statement.

`SHARED FRAME` *frame*

Defines and identifies a frame that was created by another procedure that used the DEFINE NEW SHARED FRAME statement. When you use the DEFINE SHARED FRAME statement, you cannot name any fields or variables in that frame that are not already named in the frame described by the DEFINE NEW SHARED FRAME statement.

[PRIVATE] FRAME *frame*

Defines and identifies a frame widget as a class-scoped object. A class-scoped handle-based object is not a member of a class, but provides a resource that is privately available to the class definition similar to a non-shared data element in a procedure definition. The option to specify the PRIVATE access mode is provided for readability. You cannot specify PRIVATE when defining a frame widget as a data element in a method or procedure.

Note: This option is applicable only when defining a class-scoped frame widget in a class definition (`.cls`) file.

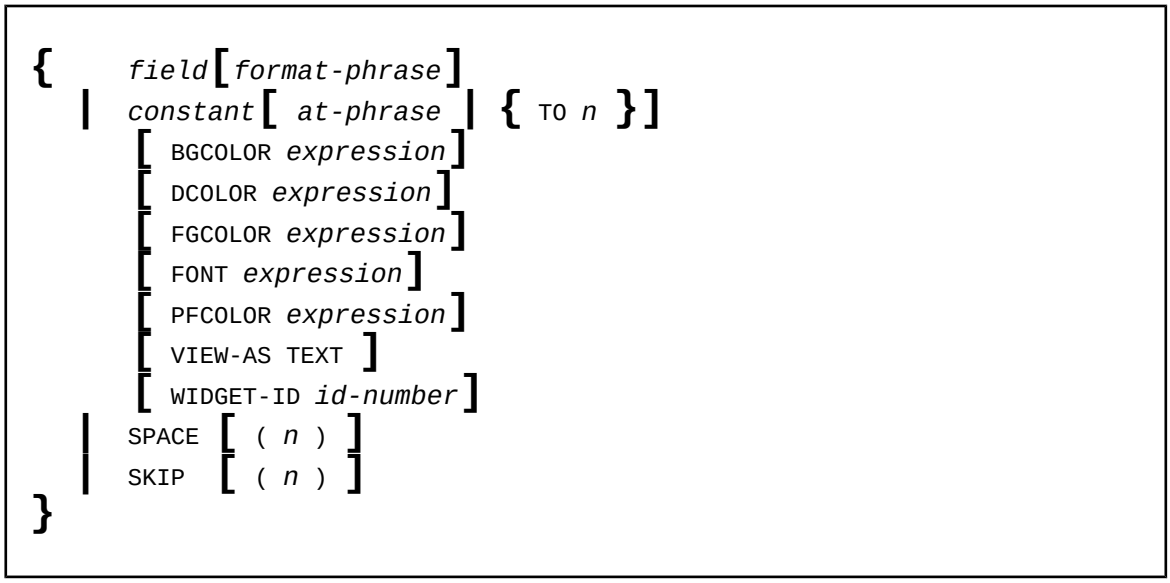
FRAME *frame*

Defines and identifies a frame widget for access only within the current procedure, class, or method of a class.

form-item

Specifies a field-level widget or value to display in the frame, or a SPACE or SKIP directive. The data specified by all form items is owned by a single field group, duplicated for each data iteration in the frame.

This is the syntax for *form-item*:



field

Specifies a field-level widget to be displayed in the frame. This value cannot be an expression or a frame. To specify a child frame, you must first define the parent and child frames, and then assign the FRAME attribute of the child frame to the handle of the parent frame. The child frame is assigned to the same field group as other form items.

format-phrase

Specifies one or more frame attributes for a field or variable. For more information on *format-phrase*, see the Format phrase reference entry.

constant

A constant value.

at-phrase

Specifies the location of a value within the frame. The AT phrase does not left justify the data; it simply indicates the placement of the data area. This is the syntax for *at-phrase*:

```
AT { COLUMN column | COLUMN-OF reference-point }
  { ROW row | ROW-OF reference-point }
    [ COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED ]
```

```
AT { X x | X-OF reference-point }
  { Y y | Y-OF reference-point }
  [ COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED ]
```

```
AT n
```

For more information, see the AT phrase reference entry.

TO *n*

The number (*n*) of the column where you want the right edge of the value. The TO option does not right justify the data; it simply indicates the placement of the data area.

BGCOLOR *expression*

Specifies the background color of the form item in graphical interfaces. This option is ignored in character interfaces.

DCOLOR *expression*

Specifies the display color of the form item in character interfaces. This option is ignored in graphical interfaces.

FGCOLOR *expression*

Specifies the foreground color of the form item in graphical interfaces. This option is ignored in character interfaces.

FONT *expression*

Specifies the font of the form item.

PFCOLOR *expression*

Specifies the prompt color of the form item in character interfaces. This option is ignored in graphical interfaces.

VIEW-AS TEXT

Specifies that the form item displays as a TEXT widget rather than as a FILL-IN.

WIDGET-ID *id-number*

Specifies a widget ID for a field-level widget or value to display in a frame. The value of *id-number* must be an expression that evaluates to an even integer value between 2 and 65534, inclusive, and must be unique across all widget IDs in the window or dialog box.

If you specify an invalid ID, the compiler displays an error message. This option is supported in graphical interfaces only, and only in Windows.

SPACE (*n*)

Identifies the number (*n*) of blank spaces to insert after the expression displays. The *n* can be 0. If the number of spaces you specify is more than the spaces left on the current line of the frame, the AVM starts a new line and discards extra spaces. If you do not use this option or you do not use *n*, the AVM inserts one space between items in the frame.

SKIP (*n*)

Identifies the number (*n*) of blank lines to insert after the expression is displayed. The number of blank lines can be 0. If you do not use this option, the AVM does not skip a line between expressions unless the expressions do not fit on one line. If you use the SKIP option but do not specify *n*, or if *n* is 0, the AVM starts a new line unless it is already at the beginning of a new line.

record

Represents the name of the record you want to display. Naming a record is shorthand for listing each field individually, as a form item.

EXCEPT *field* . . .

Tells the AVM to display all the fields in the frame except those fields listed in the EXCEPT phrase.

HEADER

Tells the AVM to place the following items in a header section at the top of the frame in a separate field group from all other data. In addition to fields, variables, and constants, the frame header can contain expressions, images, and rectangles. The AVM reevaluates these expressions each time it displays the frame.

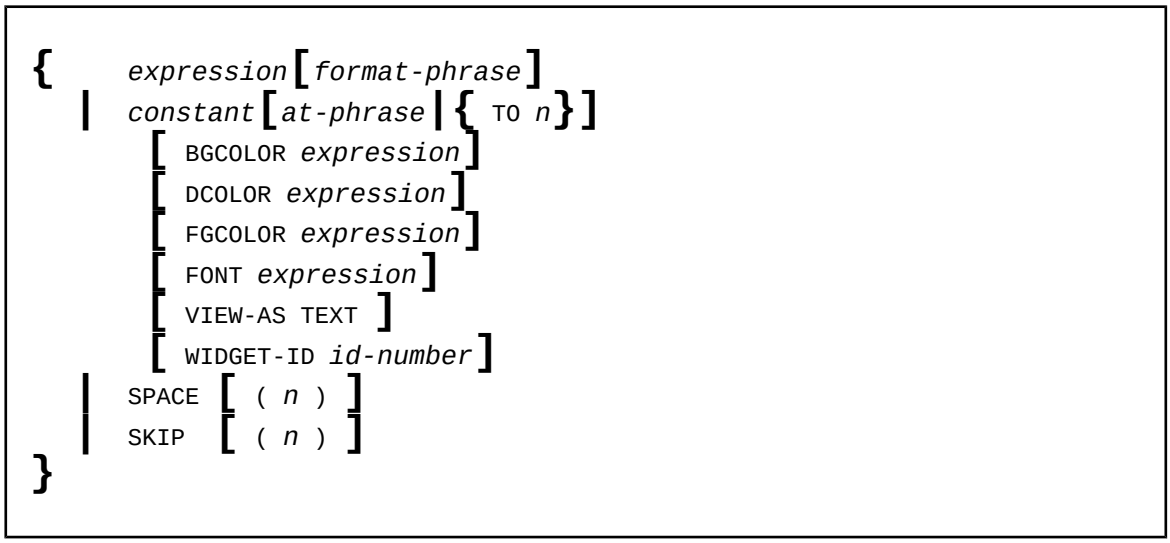
When you use the HEADER option, the AVM disregards OpenEdge Data Dictionary field labels for fields you name in the DEFINE FRAME statement. Use character strings to specify labels on fields you name in the frame header.

BACKGROUND

Specifies that any following frame items are displayed in the frame background, behind the data and header in a separate field group. Typically, this option is used to display images or rectangles behind the data.

head-item

A description of a value displayed in the frame header or background, or a SPACE or SKIP directive. Following is the syntax for *head-item*:



This is exactly the same as the syntax for a *form-item*, except that a *head-item* can be an expression and does not include the PFCOLOR option. If you use an expression in a HEADER or BACKGROUND phrase, the expression is evaluated each time the frame is viewed. If you give the PAGE-TOP or PAGE-BOTTOM option for the frame, the expression is evaluated for each page. This lets you include a reference to the PAGE-NUMBER function in the frame header.

frame-phrase

Specifies additional options for the frame, including the VIEW-AS DIALOG-BOX option to define the frame as a dialog box. For more information on frame and dialog box options, see the Frame phrase reference entry.

Examples

The following example, `r-deffrm.p`, uses the DEFINE FRAME statement to set up the format of a frame. It then scopes that frame to a FOR EACH block.

r-deffrm.p

```
DEFINE VARIABLE bal-avail NO-UNDO LIKE Customer.Balance
  COLUMN-LABEL "Available!Credit" NO-UNDO.

DEFINE FRAME cust-bal
  Customer.CustNum
  Customer.Name FORMAT "X(20)"
  Customer.CreditLimit LABEL "Limit"
  Customer.Balance
  bal-avail
  WITH CENTERED ROW 3 TITLE "Available Customer Credit" USE-TEXT.

FOR EACH Customer NO-LOCK WITH FRAME cust-bal:
  DISPLAY
    Customer.CustNum
    Customer.Name
    Customer.CreditLimit
    Customer.Balance
    Customer.CreditLimit - Customer.Balance @ bal-avail.
END.
```

The following example defines three frames. The cust-info frame is scoped to the trigger for the b_next button where it is first referenced. Similarly, the `cust-dtl` frame is scoped to the `b_dtl` trigger. The `butt-frame` frame is scoped to the outer procedure block.

r-dffrm1.p

```
DEFINE BUTTON b_dtl  LABEL "Detail".
DEFINE BUTTON b_next LABEL "Next".
DEFINE BUTTON b_quit LABEL "Quit" AUTO-ENDKEY.

DEFINE FRAME cust-info
  Customer.CustNum
  Customer.Name FORMAT "X(20)"
  Customer.Phone
  WITH CENTERED ROW 4.

DEFINE FRAME cust-dtl
  Customer EXCEPT Customer.CustNum Customer.Name Customer.Phone
  WITH SIDE-LABELS ROW 9 WIDTH 100.

DEFINE FRAME butt-frame
  b_dtl b_next b_quit
  WITH ROW 1.

ON CHOOSE OF b_dtl
  DISPLAY Customer EXCEPT Customer.CustNum Customer.Name Customer.Phone
    WITH FRAME cust-dtl.

ON CHOOSE OF b_next DO:
  HIDE FRAME cust-dtl.
  FIND NEXT Customer NO-LOCK NO-ERROR.
  IF NOT AVAILABLE Customer THEN
    FIND LAST Customer NO-LOCK.

  DISPLAY Customer.CustNum Customer.Name Customer.Phone
    WITH FRAME cust-info.
END.

ENABLE ALL WITH FRAME butt-frame.
APPLY "CHOOSE" TO b_next IN FRAME butt-frame.
WAIT-FOR CHOOSE OF b_quit.
```

The following example uses a set of thin rectangles as lines to create graphic columns within a frame background:

r-bkgrnd.p

```
DEFINE VARIABLE item-tot AS DECIMAL NO-UNDO LABEL "Value".

DEFINE RECTANGLE vline1 SIZE .4 BY 5 EDGE-PIXELS 2.
DEFINE RECTANGLE vline2 LIKE vline1.
DEFINE RECTANGLE vline3 LIKE vline1.
DEFINE RECTANGLE vline4 LIKE vline1.
DEFINE RECTANGLE vline5 LIKE vline1.
DEFINE RECTANGLE vline6 LIKE vline1.

DEFINE RECTANGLE hline SIZE 78 BY .1 EDGE-PIXELS 2.

DEFINE FRAME item-info
  Item.ItemNum
  Item.ItemName
  Item.OnHand
  Item.ReOrder
  Item.OnOrder
  Item.Price
  Item-tot
  BACKGROUND SKIP(1) hline
    vline1 AT 9
    vline2 AT 25
    vline3 AT 33
    vline4 AT 42
    vline5 AT 51
    vline6 AT 65
  WITH TITLE "Inventory Current Value" CENTERED USE-TEXT 5 DOWN.

FOR EACH Item NO-LOCK WITH FRAME item-info:
  DISPLAY
    Item.ItemNum
    Item.ItemName
    Item.OnHand
    Item.ReOrder
    Item.OnOrder
    Item.Price
    Item.OnHand * Item.Price @ item-tot.
```

The following procedure defines the shared frame cust-frame. It also defines a shared variable and a shared buffer. For each Customer whose Customer number is less than 20, the procedure displays Customer information in the cust-frame. The format for the cust-frame is defined in the `r-shrfrm.i` include file.

r-shrfrm.p

```
DEFINE NEW SHARED FRAME cust-frame.
DEFINE NEW SHARED VARIABLE csz AS CHARACTER FORMAT "x(29)".
DEFINE NEW SHARED BUFFER xcust FOR Customer.

FOR EACH xcust WHERE xcust.CustNum <=20:
  {r-shrfrm.i}    /* shared frame layout */

  DISPLAY xcust.Name xcust.Phone xcust.Address xcust.SalesRep
    xcust.City + ", " + xcust.State + " " + xcust.PostalCode @ csz
    xcust.CreditLimit WITH FRAME cust-frame.

  RUN r-updord.p. /* Update Customer's orders */
END.
```

Following is the `r-shrfrm.i` include file:

r-shrfrm.i

```
FORM
  xcust.Name COLON 10
  xcust.Phone COLON 55
  xcust.Address COLON 1
  xcust.SalesRep COLON 55
  csz NO-LABEL COLON 10
  xcust.CreditLimit COLON 55 SKIP(2)
  Order.OrderNum COLON 10 Order.OrderDate COLON 30
  Order.ShipDate COLON 30
  Order.PromiseDate COLON 30
  WITH SIDE-LABELS 1 DOWN CENTERED ROW 5
  TITLE "Customer/Order Form" FRAME cust-frame.
```

After the `r-shrfrm.p` procedure displays the Customer information, it calls the `r-updord.p` procedure.

The `r-updord.p` procedure defines the variable, frame, and buffer that were originally defined in the `r-shrfrm.p` procedure. However, in this second reference to the items, the keyword NEW is omitted. The `r-updord.p` procedure displays, and lets you update, the Order information for the Customer displayed in the cust-frame. The Order information is displayed in the same frame.

r-updord.p

```
DEFINE SHARED FRAME cust-frame.
DEFINE SHARED VARIABLE csz AS CHARACTER NO-UNDO FORMAT "x(29)".
DEFINE SHARED BUFFER xcust FOR Customer.

FOR EACH Order OF xcust:
  {r-shrfrm.i }  /* shared frame layout */

  DISPLAY Order.OrderNum WITH FRAME cust-frame.
  UPDATE Order.OrderDate Order.ShipDate Order.PromiseDate
    WITH FRAME cust-frame.
END.
```

The following example, `r-fof1.p`, creates a dialog box to display Customer information from a query. The dialog box contains three child frames to display Customer contact information (FRAME cont-fr), Customer account information (FRAME acct-fr), and control buttons for moving through the query results list (FRAME `ctrl-fr`).

r-fof1.p

```
DEFINE QUERY custq FOR Customer.
DEFINE BUTTON bprev LABEL "<".
DEFINE BUTTON bnext LABEL ">".
DEFINE FRAME cust-fr SKIP(.5)
 SPACE(8) Customer.Name Customer.CustNum Customer.SalesRep
 Customer.Comments AT COLUMN 6 ROW 13.5
 WITH SIDE-LABELS TITLE "Customer Data" SIZE 80 BY 15 VIEW-AS DIALOG-BOX.
DEFINE FRAME cont-fr SKIP(.5)
 Customer.Address COLON 17 SKIP
 Customer.Address2 COLON 17 SKIP
 Customer.City COLON 17 SKIP
 Customer.State COLON 17 SKIP
 Customer.PostalCode COLON 17 SKIP
 Customer.Country COLON 17 SKIP
 Customer.Contact COLON 17 SKIP
 Customer.Phone COLON 17
 WITH SIDE-LABELS TITLE "Contact Informaion" SIZE 40 BY 10 AT COLUMN 1 ROW 3.
```

```
DEFINE FRAME ctrl-fr SKIP(.12)
 SPACE(4) bprev bnext
 WITH TITLE "PREVIOUS/NEXT" SIZE 15 BY 2 AT COLUMN 53 ROW 10.5.
DEFINE FRAME acct-fr SKIP(.5)
 Customer.Balance COLON 15 SKIP
 Customer.CreditLimit COLON 15 SKIP
 Customer.Discount COLON 15 SKIP
 Customer.Terms COLON 15
 WITH SIDE-LABELS TITLE "Account Information"
 SIZE 38.85 BY 6 AT COLUMN 41 ROW 3.

ON CHOOSE OF bnext DO:
  GET NEXT custq.
  IF NOT AVAILABLE Customer THEN GET FIRST custq.
  RUN display-proc IN THIS-PROCEDURE.
END.

ON CHOOSE OF bprev DO:
  GET PREV custq.
  IF NOT AVAILABLE Customer THEN GET LAST custq.
  RUN display-proc IN THIS-PROCEDURE.
END.

FRAME cont-fr:FRAME = FRAME cust-fr:HANDLE.
FRAME acct-fr:FRAME = FRAME cust-fr:HANDLE.
FRAME ctrl-fr:FRAME = FRAME cust-fr:HANDLE.

OPEN QUERY custq PRESELECT EACH Customer BY Customer.Name.
GET FIRST custq.
RUN display-proc IN THIS-PROCEDURE.
ENABLE ALL WITH FRAME ctrl-fr.

WAIT-FOR WINDOW-CLOSE OF FRAME cust-fr.

PROCEDURE display-proc:
  DISPLAY
    Customer.Name Customer.CustNum Customer.SalesRep
    Customer.Comments WITH FRAME cust-fr.
  DISPLAY
    Customer.Address Customer.Address2 Customer.City Customer.State
    Customer.PostalCode Customer.Country Customer.Contact Customer.Phone
    WITH FRAME cont-fr.
  DISPLAY
    Customer.Balance Customer.CreditLimit Customer.Discount Customer.Terms
    WITH FRAME acct-fr.
END PROCEDURE.
```

Notes

- You cannot define a SHARED or NEW SHARED frame widget in a persistent procedure. If you do, ABL raises ERROR on the RUN statement that creates the procedure.
- You cannot define a SHARED or NEW SHARED frame widget in a class definition (`.cls`) file. If you do, ABL generates a compilation error.
- If you do not specify the font for a frame, ABL uses the system default font, not the font of the window. This is because ABL determines the frame layout at compile time when the window's fonts (known at run time) are not yet available.
- You can use just one DEFINE FRAME statement per frame in a procedure.
- If you name variables or parent child frames to a shared frame, ABL does not automatically make those variables and child frames shared. If you want to share the variables and child frames among procedures, you must define each variable and frame using the SHARED option in all the sharing procedures.

- ABL scopes a newly defined frame to the block that first references the frame. (The DEFINE FRAME statement does not count as a reference.) ABL scopes a shared frame outside of the called procedure.
- The *frame-phrase* options specified in a DEFINE NEW SHARED FRAME statement are carried over to all corresponding DEFINE SHARED FRAME statements and cannot be overridden.
- You can use different field-level help and validation in new shared, and shared frames.
- You must define a shared frame before referencing that frame in a procedure.
- All frame fields and Frame phrase options in a shared frame must first be defined in the initial DEFINE NEW SHARED FRAME statement or an additional FORM statement in the same procedure. Procedures that share this frame only have to define fields that correspond to the fields in the initial definition plus any specified ACCUM option. Other Frame phrase options for the SHARED frames are allowed, but are ignored except for the ACCUM option. This allows you to make use of the same FORM statement in an include file for both the NEW SHARED and matching SHARED frames. See the FORM statement reference entry for more information.
- If you use an Aggregate phrase to accumulate a value within a shared frame, you must also use the ACCUM option in each procedure that uses the shared frame.
- If you define a frame to use as a DDE frame, you must realize the frame (display it) before using it as a conversation end-point. If you want the DDE frame to remain invisible during its use in a DDE conversation, set its HIDDEN attribute to TRUE after realizing the frame. For information on DDE frames, see *OpenEdge Programming Interfaces*.
- If you have enabled application-defined widget IDs in your ABL GUI application, by specifying the Use Widget ID (`-usewidgetid`) startup parameter, then the AVM uses the value specified in the WIDGET-ID option to set the WIDGET-ID attribute for this widget when it creates the widget at run time, instead of using the widget ID it normally generates by default. If you have not enabled application-defined widget IDs, then the AVM ignores this option setting at run time.

  For more information about the WIDGET-ID attribute, see its reference entry in the Handle Attributes and Methods Reference. For more information about the Use Widget ID (`-usewidgetid`) startup parameter, see *Startup Command and Parameter Reference*.

See also

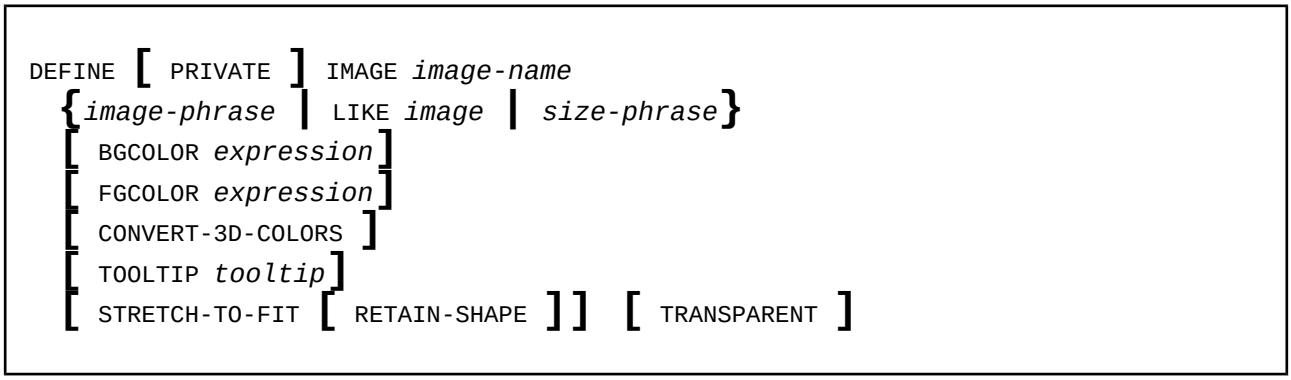
Class-based data member access, DEFINE BUFFER statement, DEFINE VARIABLE statement, FORM statement, Frame phrase, RUN statement

DEFINE IMAGE statement
(Windows only; Graphical interfaces only)

Defines an image widget in a graphical interface that is created at compile time for use within a single procedure or class. An image widget is a container for an operating system image file and can be displayed in a form or used as a form background.

Note: Does not apply to SpeedScript programming.

Syntax



```
DEFINE [ PRIVATE ] IMAGE image-name
  {image-phrase | LIKE image | size-phrase}
  [ BGCOLOR expression ]
  [ FGCOLOR expression ]
  [ CONVERT-3D-COLORS ]
  [ TOOLTIP tooltip ]
  [ STRETCH-TO-FIT [ RETAIN-SHAPE ] ] [ TRANSPARENT ]
```

`[ PRIVATE ] IMAGE image-name`

Defines and identifies an image widget as a class-scoped object. A class-scoped handle-based object is not a member of a class, but provides a resource that is privately available to the class definition similar to a non-shared data element in a procedure definition. The option to specify the PRIVATE access mode is provided for readability. You cannot specify PRIVATE when defining an image widget as a data element in a method or procedure.

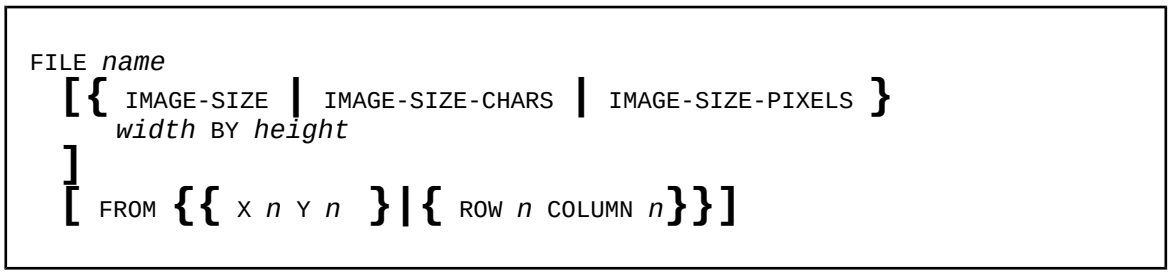
Note: This option is applicable only when defining a class-scoped image widget in a class definition (`.cls`) file.

`IMAGE image-name`

Defines and identifies an image widget for access only within the current procedure, class, or method of a class.

`image-phrase`

Specifies the file where the image is stored and the portion of the image to read. This is the syntax for image-phrase:



```
FILE name
  [{ IMAGE-SIZE | IMAGE-SIZE-CHARS | IMAGE-SIZE-PIXELS }
     width BY height
  ]
  [ FROM {{ X n Y n } | { ROW n COLUMN n }} ]
```

For more information on this syntax, see the Image phrase reference entry.

You must specify either the LIKE option, an Image phrase or a Size phrase within the DEFINE IMAGE statement, and you may specify any two or all three.

`LIKE image`

Specifies a previously defined image from which this image inherits attributes. You can override specific attributes by specifying other options of the DEFINE IMAGE statement.

You must specify either the LIKE option, an Image phrase or a Size phrase within the DEFINE IMAGE statement, and you may specify any two or all three.

size-phrase

Specifies the outside dimensions of the image widget. This is the syntax for size-phrase:

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS }width BY height
```

If you specify SIZE or SIZE-CHARS, the units are characters; if you specify SIZE-PIXELS, the units are pixels. If you use character units, the values *width* and *height* must be decimal constants; for pixel units, they must be integer constants. For more information, see the SIZE phrase reference entry.

You must specify either the LIKE option, an Image phrase or a Size phrase within the DEFINE IMAGE statement, and you may specify any two or all three.

BGCOLOR *expression*

Has no effect; supported only for backward compatibility.

FGCOLOR *expression*

Has no effect; supported only for backward compatibility.

CONVERT-3D-COLORS

Specifies that the colors associated with an image will be converted to the system 3D colors when an image is loaded. The following table describes the color conversion process.

Table 30: 3D-color conversions for images

If the color is . . .	And the original Red-Green-Blue (RGB) color value is . . .	Then the new converted system color is . . .
White	(255, 255, 255)	System button highlight color
Light Gray	(192, 192, 192)	System button face color
Dark Gray	(128, 128, 128)	System button shadow color
Black	(0, 0, 0)	System button text color

During a session, if Windows notifies the AVM that the system colors are changed, all images that have this option are reloaded and converted to the new system colors.

TOOLTIP *tooltip*

Allows you to define a help text message for an image widget. The AVM automatically displays this text when the user pauses the mouse pointer over the image widget.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the ToolTip is removed from the button. No ToolTip is the default. ToolTips are supported in Windows only.

STRETCH-TO-FIT

Forces the image to expand or contract to fit within the image widget's boundaries.

This option has no effect if an icon is displayed on the image widget.

RETAIN-SHAPE

Indicates that the image should retain its aspect ratio (expand or contract equally in both dimensions). This may leave some uncovered space at the bottom or right of the image widget.

RETAIN-SHAPE is ignored if STRETCH-TO-FIT is FALSE or if an icon is displayed on the image widget.

TRANSPARENT

Indicates that the background color of the image is transparent. The background color is determined by the color of the pixel in the lower left corner of the image.

The TRANSPARENT option overrides the CONVERT-3D-COLORS option; if both are set, CONVERT-3D-COLORS is ignored.

This option has no effect if an icon is displayed on the image widget.

Example

This procedure defines an image widget named `trashcan`, and loads into the widget a series of operating system image files that create an animation of a fire burning in a trash can. The user begins the animation by choosing the Animate button. This procedure serves as an illustration only. It depends on the existence of image filenames `ANI01`, `ANI02`. ... `ANI14` , which are not provided.

r-image.p

```
DEFINE VARIABLE repeat_loop    AS INTEGER NO-UNDO.
DEFINE VARIABLE animation_loop AS INTEGER NO-UNDO..
DEFINE VARIABLE lok            AS LOGICAL NO-UNDO..

DEFINE BUTTON animate LABEL "Animate".
DEFINE IMAGE trashcan FILE "ANI01.BMP".

DISPLAY animate trashcan WITH FRAME y TITLE "**  Animation Sample  **".

ON CHOOSE OF animate IN FRAME y DO:
  /* Begin Animation */
  DO repeat_loop = 1 TO 5:
    DO animation_loop = 1 TO 14:
      lok = trashcan:LOAD-IMAGE("ANI" + STRING(animation_loop,"99"))
        IN FRAME y.
    END.
  END.
END.
UPDATE animate WITH FRAME y.
```

Notes

- In Windows, if the file has no extension, the AVM by default looks for image files with either a `.bmp` or `.ico` extension.
- To create the compile-time defined image you are defining, you must define a compile-time defined frame that contains the image. Each frame you define that contains the same image creates an additional instance of that image. The handle for a compile-time defined image is not available until the image is created.

- The AVM only performs the color conversion process on bitmaps (`.bmp` files) that contain 256 or fewer colors. However, you might consider using 16-color bitmaps because only the first sixteen entries in the bitmap's color table will be converted.
- Icon colors (`.ico` files) are not converted, even if CONVERT-3D-COLORS is TRUE.
- See Image phrase for the list of supported image file formats.
- You can specify an application-defined widget ID for a compile-time defined image using the *form-item* phrase in either the FORM statement or the DEFINE FRAME statement. See the FORM statement and DEFINE FRAME statement reference entries for more information.

See also

Class-based data member access, FORM statement, Image phrase

DEFINE MENU statement

Defines a menu widget that is created at compile time for use in one or more procedures, or within a single class. You can use a menu widget as a pop-up menu or a menu bar.

Note: Does not apply to SpeedScript programming.

Syntax

```
DEFINE { [ [ NEW ] SHARED ] | [ PRIVATE ] } MENU menu-name
   [ FGCOLOR expression ]
   [ BGCOLOR expression ]
   [ DCOLOR expression ]
   [ PFCOLOR expression ]
   [ FONT number ]
   [ { TITLE title } | MENUBAR ]
   [ { LIKE menu } | menu-element-descriptor . . . ]
```

`NEW SHARED MENU` *menu-name*

Defines and identifies a menu widget that can be used by other procedures. The menu remains available to other procedures until the procedure that contains this statement ends.

`SHARED MENU` *menu-name*

Defines and identifies a menu widget that was created in another procedure with the DEFINE NEW SHARED MENU statement.

[PRIVATE] MENU *menu-name*

Defines and identifies a menu widget as a class-scoped object. A class-scoped handle-based object is not a member of a class, but provides a resource that is privately available to the class definition similar to a non-shared data element in a procedure definition. The option to specify the PRIVATE access mode is provided for readability. You cannot specify PRIVATE when defining a menu widget as a data element in a method or procedure.

Note: This option is applicable only when defining a class-scoped menu widget in a class definition (`.cls`) file.

MENU *menu-name*

Defines and identifies a menu widget for access only within the current procedure, class, or method of a class.

BGCOLOR *expression*

Supported only for backward compatibility.

DCOLOR *expression*

Specifies the display color for the menu in character interfaces. This option is ignored in graphical interfaces.

FGCOLOR *expression*

Supported only for backward compatibility.

PFCOLOR *expression*

Specifies the prompt-for color for the menu in character interfaces. This option is ignored in graphical interfaces.

FONT *number*

Supported only for backward compatibility.

MENUBAR

Specifies that the menu displays as a menu bar.

TITLE *title*

Specifies the title of the menu. Only pop-up menus can have titles. This option is invalid for menu bars. The title displays at the top of the menu. In environments that do not support this option, it is ignored.

LIKE *menu*

Specifies a previously defined menu whose characteristics you want to apply to the new menu. If you name a menu with this option, you must have defined that menu previously in the procedure.

menu-element-descriptor

Specifies an element display on the menu. Each element is either a normal menu item, a submenu, a rule, or a blank space. The last two are valid only for pop-up menus. You must specify one or more menu elements, unless you use the LIKE option.

This is the syntax for *menu-element-descriptor*:

```
{ menu-item-phrase
  SUB-MENU submenu [ DISABLED ] [ LABEL label ]
  RULE
  SKIP
}
```

RULE

Specifies that a rule or line is inserted at this point in the menu. You can use this, for example, to divide the menu into sections.

SKIP

Specifies that a blank line is inserted at this point in the menu. You can use this, for example, to divide the menu into sections.

SUB-MENU *submenu* [DISABLED] [LABEL *label*]

Specifies that a submenu displays as a menu item. The submenu must be previously defined in the procedure. The submenu appears when the user chooses that item. The submenu cannot be a menu bar. The DISABLED and LABEL options for a submenu are the same as described for the *menu-item-phrase*.

menu-item-phrase

Specifies a normal menu item. This is the syntax for *menu-item-phrase*:

```
MENU-ITEM menu-item-name
  [ ACCELERATOR keylabel ]
  [ BGCOLOR expression ]
  [ DCOLOR expression ]
  [ DISABLED ]
  [ FGCOLOR expression ]
  [ FONT expression ]
  [ LABEL label ]
  [ PFCOLOR expression ]
  [ READ-ONLY ]
  [ TOGGLE-BOX ]
  [ trigger-phrase ]
```

MENU-ITEM *menu-item-name*

The name of the menu item you are defining.

ACCELERATOR *keylabel*

Specifies a keyboard accelerator for this menu item. A keyboard accelerator is a key—sometimes modified by **SHIFT**, **CONTROL**, or **ALT**—that chooses a menu item even if the menu is not displayed. The value *keylabel* must be character-string expression that evaluates to a valid key label recognized by the AVM, such as a, **F1**, or **ALT+SHIFT+F1**. See the topics on handling user input in the *OpenEdge Programming Interfaces* for the precedence rules that the AVM uses for handling keyboard input.

BGCOLOR *expression*

Specifies the background color for the menu item in graphical environments. If you omit this option, the menu item inherits the background color of the menu.

DCOLOR *expression*

Specifies the display color for the menu item in character interfaces. If you omit this option, the menu item inherits the display color of the menu.

DISABLED

Specifies that the menu item is initially disabled for input. This means that the user cannot choose this item. Disabled items are grayed out (in environments that support it).

FGCOLOR *expression*

Specifies the foreground color for the menu item in graphical environments. If you omit this option, the menu item inherits the foreground color of the menu.

FONT *expression*

Specifies the font for the menu item. If you omit this option, the menu item inherits the font of the menu.

LABEL *label*

Specifies the text that is displayed in the menu for a choosable menu item or submenu. Include an ampersand (&) within the label to assign the following letter as a mnemonic for the menu item. This means that when the menu is displayed, the user can choose the item by pressing that single key. If you do not include an ampersand within the label, Windows treats the first character as a mnemonic.

To include a literal ampersand within a label, specify two ampersands (&&).

PFCOLOR *expression*

Specifies the prompt-for color for the menu item in character interfaces. If you omit this option, the menu item inherits the prompt-for color of the menu.

READ-ONLY

Specifies that this menu item is read-only text. The user cannot choose this item.

`TOGGLE-BOX`

Specifies that the menu item is displayed as a checkbox that the user can toggle on or off. In environments that do not support this option, it is ignored.

`trigger-phrase`

Specifies application triggers for the menu item. Typically, you associate a CHOOSE trigger with each menu item.

For more information, see the Trigger phrase reference entry.

Example

The `r-bar.p` procedure defines a menu bar, mbar, that contains three pull-down submenus labeled Topic, Move, and Exit. The handle of mbar is assigned to the current window. The ON statements define triggers to execute when you choose the corresponding menu items.

r-bar.p

```
DEFINE SUB-MENU topic
  MENU-ITEM numbr    LABEL "Cust. Number"
  MENU-ITEM addr     LABEL "Address"
  MENU-ITEM othrinfo LABEL "Other".
DEFINE SUB-MENU move
  MENU-ITEM forward  LABEL "NextRec" ACCELERATOR "PAGE-DOWN"
  MENU-ITEM backward LABEL "PrevRec" ACCELERATOR "PAGE-UP".
DEFINE SUB-MENU quitit
  MENU-ITEM quititem LABEL "E&xit".

DEFINE MENU mbar MENUBAR
  SUB-MENU topic  LABEL "Topic"
  SUB-MENU move   LABEL "Move"
  SUB-MENU quitit LABEL "E&xit".

ON CHOOSE OF MENU-ITEM numbr
  DISPLAY Customer.CustNum.

ON CHOOSE OF MENU-ITEM addr
  DISPLAY Customer.Address Customer.Address2 Customer.City
    Customer.State Customer.PostalCode
    WITH FRAME addr-frame NO-LABELS COLUMN 25.

ON CHOOSE OF MENU-ITEM othrinfo
  DISPLAY Customer EXCEPT Customer.Name Customer.CustNum Customer.Address
    Customer.Address2 Customer.City Customer.State Customer.PostalCode
    WITH FRAME oth-frame SIDE-LABELS.

ON CHOOSE OF MENU-ITEM forward DO:
  HIDE ALL NO-PAUSE.
  CLEAR FRAME name-frame.
  FIND NEXT Customer NO-ERROR.
  IF AVAILABLE Customer THEN
    DISPLAY Customer.Name WITH FRAME name-frame.
END.

ON CHOOSE OF MENU-ITEM backward DO:
  HIDE ALL NO-PAUSE.
  CLEAR FRAME name-frame.
  FIND PREV Customer NO-ERROR.
  IF AVAILABLE Customer THEN
    DISPLAY Customer.Name WITH FRAME name-frame.
END.

FIND FIRST Customer.
DISPLAY Customer.Name LABEL "Customer Name" WITH FRAME name-frame.
ASSIGN CURRENT-WINDOW:MENUBAR = MENU mbar:HANDLE.
WAIT-FOR CHOOSE OF MENU-ITEM quititem.
```

Notes

- You cannot define a SHARED or NEW SHARED menu widget in a persistent procedure. If you do, ABL raises ERROR on the RUN statement that creates the procedure.
- You cannot define a SHARED or NEW SHARED menu widget in a class definition (`.cls`) file. If you do, ABL generates a compilation error.
- Keyboard accelerators are specified for menu-items forward and backward. The user can press PAGE-DOWN key to look at the next Customer record and the PAGE-UP to view the previous Customer record.
- The menu item quititem has a label E&xit; the ampersand makes X the mnemonic for that menu item.

- You cannot define a submenu with the same name more than once in the same menu tree. Thus, if menu mFile contains both submenu mOptions and submenu mSave, submenu mSave cannot also contain submenu mOptions.
- Menu items in different menus and submenus can have the same names. In the above procedure, the menu items in myfile and myobjects share the same names. To avoid ambiguity, use the IN MENU or IN SUB-MENU option to identify the parent menu or submenu.
- There are instances where you cannot avoid ambiguity in menu item references. In such instances, ABL always references the first unambiguous instance of the menu item. In particular, if the same submenu containing a menu item appears in more than one menu and each menu defines another instance of the same menu item, you can only reference that menu item in the submenu from the first menu that contains it. Thus, if submenu mOptions contains menu item mSave and the menus mFile and mDraw (in that order) both contain submenu mOptions and another menu item mSave, you can only reference menu item mSave in submenu mOptions from menu mFile. You cannot uniquely reference menu item mSave in submenu mOptions from menu mDraw because menu mDraw contains another menu item mSave.

See also

Class-based data member access, COLOR phrase, DEFINE SUB-MENU statement, RUN statement

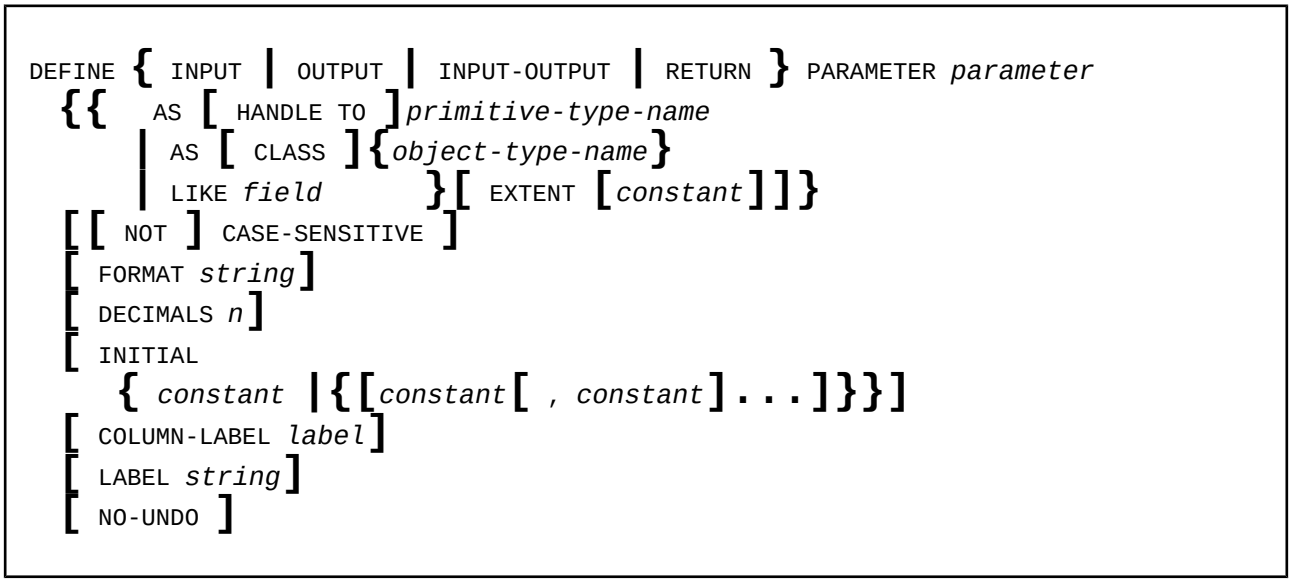
DEFINE PARAMETER statement

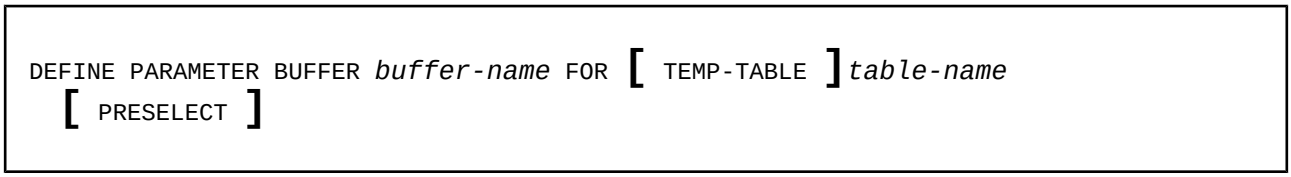
Defines a run-time parameter in an ABL procedure (internal or external), Windows dynamic link library (DLL) routine, UNIX shared library routine, or ActiveX control event procedure.

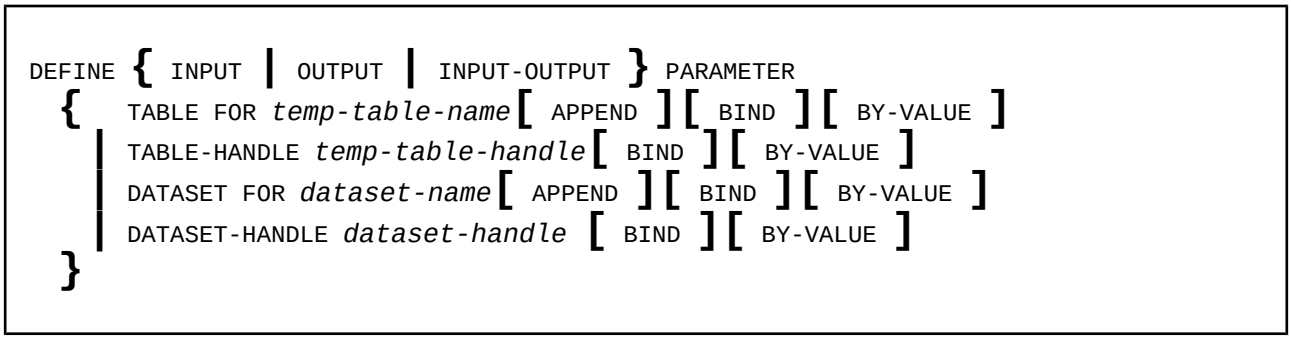
Note: To define run-time parameters of a user-defined function, or a method within a class (including constructors), see the Parameter definition syntax reference entry.

Each parameter requires its own DEFINE statement. The parameters must be specified in the RUN statement in the same order they are defined with DEFINE statements. In addition, the parameter types (INPUT, OUTPUT, INPUT-OUTPUT, RETURN, TABLE, TABLE-HANDLE, DATASET, DATASET-HANDLE, and BUFFER) specified in the DEFINE and RUN statements must agree. The corresponding data types and run-time values must also be compatible enough to allow the AVM to perform any necessary conversions.

Syntax







`INPUT PARAMETER`

Defines a parameter that gets its value from one of the following sources:

- If the calling procedure runs the current (called) procedure synchronously, the value comes from the corresponding INPUT parameter of the RUN statement.
- If the current procedure or method is specified to handle the PROCEDURE-COMPLETE callback event for an asynchronous remote procedure, the value comes from the corresponding OUTPUT or INPUT-OUTPUT parameter of the remote procedure.

`OUTPUT PARAMETER`

Defines a parameter that returns a value to one of the following destinations:

- If the calling procedure runs the current (called) procedure synchronously, the value is returned to the corresponding OUTPUT parameter of the RUN statement in the calling procedure.
- If the calling procedure runs the current (called) procedure as an asynchronous remote procedure, the value is returned to the corresponding INPUT parameter of the event procedure or method specified to handle the PROCEDURE COMPLETE event for the current procedure.

INPUT-OUTPUT PARAMETER

Defines a parameter that receives an initial value passed from the calling procedure that can be subsequently modified by the called procedure. The calling procedure cannot pass a literal value. The called procedure returns the modified value to one of the following destinations:

- If the calling procedure runs the current (called) procedure synchronously, the value is returned to the corresponding INPUT-OUTPUT parameter of the RUN statement in the calling procedure.
- If the calling procedure runs the current (called) procedure as an asynchronous remote procedure, the value is returned to the corresponding INPUT parameter of the event procedure or method specified to handle the PROCEDURE COMPLETE event for the current procedure.

RETURN PARAMETER

Defines a parameter that holds the return value of a DLL or UNIX shared library routine. When the DLL routine returns, the value of this parameter is passed back to the calling procedure. You can only have one RETURN parameter per routine.

parameter

Identifies the name of the parameter you want to define.

AS [HANDLE TO] *primitive-type-name*

Specifies a primitive type for the parameter.

For ABL procedures, *primitive-type-name* can specify any built-in primitive type used to define variables. For more information on the available primitive types, see the Data types reference entry. For more information on defining primitive type variables, see the DEFINE VARIABLE statement reference entry.

For DLL or UNIX shared library routines, *primitive-type-name* can specify an ABL DLL data type. ABL DLL data types include the built-in ABL data types CHARACTER and MEMPTR, Windows DLL-equivalent data types, and UNIX shared library data types.

The following table shows how Windows DLL and UNIX shared library data types map to ABL DLL data types.

Table 31: Data types for DLL and UNIX shared library routine parameters

Example C data type	ABL DLL parameter data type	Windows DLL and UNIX shared library data type
`char`	`BYTE`	8-bit unsigned integer
`short`	`SHORT`	16-bit signed integer
`unsigned short` `int`	`UNSIGNED-SHORT`	16-bit unsigned integer
`long` (32-bit UNIX, Win32) `int`[29]	`LONG`[30]	32-bit signed integer
`unsigned int`	`UNSIGNED-LONG`	32-bit unsigned integer

Example C data type	ABL DLL parameter data type	Windows DLL and UNIX shared library data type
`__int64` (Win32) `long long` (UNIX 32-bit) `long` (UNIX 64-bit)	`INT64`	64-bit signed integer
`float`	`FLOAT`	4-byte floating point
`double`	`DOUBLE`	8-byte floating point
`char*`	`CHARACTER`	Address (usually 32 bits)
`c-data-type`[31]	`HANDLE TO` *`parameter-data-type`*[31]	Address (usually 32 bits)
`char*`, *`output-pointer`* (which can be `char**`, `short**`, and so on), or a pointer to a structure.	`MEMPTR`	Address (usually 32 bits)

Caution: For CHARACTER parameters, the AVM always passes the routine a pointer to the character or character string value rather than the value itself. If the routine modifies the value, it can also modify the AVM memory outside the bounds of the CHARACTER value with unpredictable results. For this reason, ABL does not allow you to use OUTPUT or RETURN for CHARACTER or LONGCHAR parameters, as well as CHARACTER or LONGCHAR array parameters, and does not recommend you use INPUT-OUTPUT for CHARACTER or LONGCHAR parameters. Rather, pass the character string as a MEMPTR parameter. For more information, see *OpenEdge Programming Interfaces*.

Note: You cannot use `RETURN` for any type of array parameter.

To indicate that the DLL or UNIX shared library parameter is a pointer to a value rather than the value itself, use the HANDLE TO option. The HANDLE TO option is required when the DLL routine expects a pointer to the value. Note that the CHARACTER data type implies the HANDLE TO option, whether or not you specify it.

For ActiveX control event procedures, *`primitive-type-name`* can specify the built-in ABL data type that maps to the COM object data type of an ActiveX event parameter. The following table shows how the COM object data types for event parameters (shown as ActiveX data types) map to ABL data types.

[29] The C data type `int` generally specifies a size that depends on the operating system.

[30] To pass a `null` pointer value to a DLL routine, pass 0 using a LONG parameter. Do not use a null MEMPTR variable to pass a `null` value. However, you can pass a `null` value in one or more elements of a MEMPTR array. If this conflicts with another way to call the DLL routine, specify a second declaration for the same routine using the ORDINAL option of the PROCEDURE statement.

[31] You can use the HANDLE TO option to specify a pointer to a primitive type. Therefore, you can use the HANDLE TO option with the parameter data types (that is, BYTE, SHORT, UNSIGNED-SHORT, LONG, FLOAT, and DOUBLE) in order to specify a pointer to the respective C data types (that is, `char`, `short`, `unsigned`, `short`, `long`, `int`, `float`, and `double`). For a CHARACTER or MEMPTR parameter, it is redundant because this data type is always passed using a pointer (`char*`).

Table 32: Data types for ActiveX control event procedures

ActiveX data type[32]	ABL data type
Array	ABL array variable
Array of bytes	RAW
Boolean (2-byte integer)	LOGICAL
Currency (8-byte integer with fixed decimal point)	DECIMAL
Date	DATE DATETIME DATETIME-TZ
Decimal	DECIMAL
Double (8-byte floating point)	DECIMAL
Error Code	INTEGER
Float (Single)	DECIMAL
Integer (2-byte integer)	INTEGER
Long (4-byte integer)	INTEGER
Object (32-bit value)	COM-HANDLE
String (character string type)	CHARACTER LONGCHAR
Signed Byte	INTEGER
Signed Long (4-byte integer)	INTEGER
Signed Short (2-byte integer)	INTEGER
Signed 8-byte integer	INT64
Unsigned Byte	INTEGER
Unsigned Long (4-byte integer)	INT64
Unsigned Short (2-byte integer)	INTEGER
Unsigned 4-byte integer	INT64

[32] For more information on these data type implementations for COM objects, see *OpenEdge Programming Interfaces*.

ActiveX data type[32]	ABL data type
Unsigned 8-byte integer	DECIMAL
Variant (variable type)	<ANYTYPE>[33]

AS [CLASS] { *object-type-name* }

Defines the parameter as an object reference with the data type of a class or interface. The default value of the parameter is the Unknown value (?). You cannot assign an initial value using the INITIAL option.

object-type-name

Specifies the type name of an ABL or .NET class or interface. Specify an object type name using the syntax described in the Type-name syntax reference entry. With an appropriate USING statement, you can also specify a class or interface name alone, without the qualifying package or namespace.

You cannot directly specify the type name of a .NET mapped object type (such as System.Int32). To define a parameter that matches a .NET mapped type, you must define it as the corresponding ABL primitive type (*primitive-type-name*).

CLASS

If the specified class or interface type name conflicts with an abbreviation for a built-in primitive type name, such as INT for INTEGER, you must specify the CLASS keyword.

For a class or interface return value, ABL passes an object reference associated with the class or interface, not a class instance itself. For more information on object references, see the Class-based object reference reference entry.

LIKE, CASE SENSITIVE, FORMAT, DECIMALS, INITIAL, COLUMN-LABEL, LABEL, NO-UNDO

For descriptions of these options, see the DEFINE VARIABLE statement reference entry.

EXTENT [*constant*]

Defines the parameter as an array of data elements, where the element data type is specified by the AS *primitive-type-name* option, the LIKE *field* option, or the AS *object-type-name* option. This option can specify an array parameter as either determinate (has a defined number of elements) or indeterminate (has an undefined number of elements). To define a determinate array parameter, specify the EXTENT option with the *constant* argument. This optional argument is an integer value that represents the number of data elements in the array parameter. To define an indeterminate array parameter, specify the EXTENT option without the *constant* argument.

The EXTENT is part of the parameter data type. For more information, see the Type-name syntax reference entry.

For more information about arrays, see Work with one-dimensional arrays in *Basic ABL*.

You cannot pass an indeterminate array with no size to a COM object, DLL routine, or UNIX shared library routine.

[33] For Variant event parameters, the AppBuilder specifies <ANYTYPE> as a place holder. You must change <ANYTYPE> to the data type that most closely matches the expected value. For more information, see the available documentation on the event parameter.

If you want to define a parameter that is like an array variable or field, using the LIKE option, but if you do not want the parameter to be an array, you can use EXTENT 0 to indicate a non-array parameter.

If you are using the AS option and you do not use the EXTENT option (or you specify *constant* as 0), the parameter is not an array parameter. If you are using the LIKE *field* option and you do not use the EXTENT option, the parameter uses the extent defined for the database field you name (if any).

PARAMETER BUFFER *buffer-name* FOR [TEMP-TABLE] *table-name* [PRESELECT]

Defines a buffer parameter, where *buffer-name* is the name you specify for the buffer and *table-name* is the name of a temp-table or database table to which the buffer is attached. You can pass a buffer associated with a database table to a buffer parameter. You cannot pass a work table to a buffer parameter. A buffer parameter is always INPUT-OUTPUT. You cannot pass buffer parameters to the application server.

Use the TEMP-TABLE option to define a buffer parameter for a temp-table when the temp-table has the same name as a database table. Otherwise, ABL associates the buffer with the database table by default. Note that you can define a temp-table buffer parameter only for an internal procedure that you define in an external procedure where the temp-table specified by *table-name* is already defined.

If you use the PRESELECT option and access the buffer parameter in a DO or REPEAT block, the AVM creates an internal list of the records selected. The PRESELECT option tells the AVM to apply that internal list to the buffer you define.

TABLE FOR *temp-table-name*

Defines a temp-table parameter.

You can pass a temp-table parameter to both local and remote procedures. The AVM passes the parameter by value, by default. That is, the caller and the called routine each have their own instance of the temp-table. When you invoke the RUN statement, the AVM deep-copies the parameter from one instance to the other. The table that is copied depends on whether the parameter is INPUT, OUTPUT, or INPUT-OUTPUT. When you pass a temp-table as an INPUT parameter, the AVM replaces the receiving instance with the source instance, by default. You can also append the copied instance to the end of the receiving instance by specifying the APPEND option. For more information about the APPEND option, see the option description later in this reference entry.

When passing a temp-table parameter to a local procedure, you can override the default deep copy and pass the parameter by reference or by binding (that is, by specifying the parameter in a RUN statement using either the BY-REFERENCE or BIND option). Passing a temp-table parameter by reference or by binding allows the caller and the called routine to access the same object instance (instead of deep-copying the parameter).

Note: When you specify the BIND option in the DEFINE PARAMETER statement, you must also specify the BIND option in the RUN statement.

For more information about passing a temp-table parameter by reference or by binding, see the Parameter passing syntax reference entry. For more information about temp-table parameters, see *Develop ABL Applications*.

TABLE-HANDLE *temp-table-handle*

Defines a temp-table handle parameter.

DATASET *dataset-name*

Defines a compile-time defined ProDataSet object parameter.

You can pass a ProDataSet object parameter to both local and remote procedures. The AVM passes the parameter by value, by default. That is, the caller and the called routine each have their own instance of the object. When you invoke the RUN statement, the AVM deep-copies the parameter from one instance to the other. The ProDataSet that is copied depends on whether the parameter is INPUT, OUTPUT, or INPUT-OUTPUT. When you pass a ProDataSet as an INPUT parameter, the AVM replaces the receiving instance with the source instance, by default. You can also append the copied instance to the end of the receiving instance by specifying the APPEND option. For more information about the APPEND option, see the option description later in this reference entry.

When passing a ProDataSet object parameter to a local procedure, you can override the default deep copy and pass the parameter by reference or by binding (that is, by specifying the parameter in a RUN statement using either the BY-REFERENCE or BIND option). Passing a ProDataSet object parameter by reference or by binding allows the caller and the called routine to access the same object instance (instead of deep-copying the parameter).

Note: When you specify the BIND option in the DEFINE PARAMETER statement, you must also specify the BIND option in the RUN statement.

For more information about passing a ProDataSet object parameter by reference or by binding, see the Parameter passing syntax reference entry. For more information on ProDataSet object parameters, see *Use ProDataSets*.

DATASET-HANDLE *dataset-handle*

Defines a ProDataSet object handle parameter.

APPEND

Specifies whether or not to append the data from a source instance to the receiving instance of a passed temp-table or ProDataSet parameter. To append input parameter data, specify the APPEND option in the DEFINE PARAMETER statement. To append output parameter data, specify the APPEND option in the RUN statement.

BIND

Indicates that a TABLE, TABLE-HANDLE, DATASET, or DATASET-HANDLE parameter binds a reference-only object in one routine to an object instance defined and instantiated in another local routine.

When you define a reference-only object in the calling routine, and you want to bind that object definition to an object instance in the called routine, define the parameter by specifying the BIND option in an INPUT or INPUT-OUTPUT parameter definition. When you define a reference-only object in the called routine, and you want to bind that object definition to an object instance in the calling routine, define the parameter by specifying the BIND option in an OUTPUT parameter definition. In either case, the reference-only object definition remains bound to the object instance until the routine containing the reference-only object definition is deleted or terminates.

Caution: Do not delete the object or routine to which a reference-only object is bound, or you might be left with references to an object that no longer exists.

You can bind multiple reference-only object definitions to the same object instance. You can also bind a single reference-only object definition to the same object instance multiple times without generating an error. However, you cannot bind a single reference-only object definition to multiple object instances.

When passing one of these parameters to a remote procedure, the AVM ignores the BIND option and deep-copies the parameter based on the specified parameter mode.

For more information about passing these parameters by binding, see the Parameter passing syntax reference entry.

BY-VALUE

Specified for an INPUT, OUTPUT, or INPUT-OUTPUT TABLE, TABLE-HANDLE, DATASET, or DATASET-HANDLE parameter in a called routine, this option forces the parameter to be passed to the local routine by value, which overrides any BY-REFERENCE option in the corresponding routine invocation. For more information on BY-REFERENCE, see the Parameter passing syntax reference entry.

Examples

In the following examples, the `r-runpar.p` procedure runs a subprocedure called `r-param.p` and passes the subprocedure an INPUT parameter. The subprocedure `r-param.p` displays the INPUT parameter.

r-runpar.p

```
RUN r-param.p (INPUT 10).
```

r-param.p

```
DEFINE INPUT PARAMETER int-param AS INTEGER NO-UNDO.

DISPLAY int-param LABEL "Integer input param"
  WITH SIDE-LABELS.
```

In the following example, the `r-runpr1.p` procedure runs a subprocedure called `r-param1.p`. This example illustrates the use of multiple parameters and shows that the parameters must be passed in the proper order and must be of the same data type. Note that if you do not specify a parameter type in the RUN statement, the AVM assumes it is an input parameter.

r-runpr1.p

```
DEFINE VARIABLE new-param AS CHARACTER NO-UNDO FORMAT "x(20)".
DEFINE VARIABLE out-param AS DECIMAL   NO-UNDO.
DEFINE VARIABLE in-param  AS INTEGER   NO-UNDO INITIAL 20.

RUN r-param1.p (OUTPUT out-param, 10, OUTPUT new-param, in-param).
DISPLAY out-param LABEL "Updated YTD Sales" SKIP new-param LABEL "Status"
  WITH SIDE-LABELS.
```

r-param1.p

```
DEFINE OUTPUT PARAMETER xout-param AS DECIMAL   NO-UNDO.
DEFINE INPUT  PARAMETER newin      AS INTEGER   NO-UNDO.
DEFINE OUTPUT PARAMETER xnew-param AS CHARACTER NO-UNDO.
DEFINE INPUT  PARAMETER xin-param  AS INTEGER   NO-UNDO.

FOR EACH Customer NO-LOCK:
  xout-param = xout-param + Customer.Balance.
END.

DISPLAY xout-param LABEL "Balance" WITH SIDE-LABELS.
ASSIGN
  xout-param = xout-param + newin + xin-param
  xnew-param = "Example Complete".
```

In the following example, the `r-runpr2.p` procedure displays information from a database table and assigns the value of a database field to a variable called io-param. The variable is passed as an INPUT-OUTPUT parameter to a subprocedure called `r-param2.p`. The subprocedure `r-param2.p` performs a calculation on the INPUT-OUTPUT parameter, then passes it back to the main procedure. The `r-runpr2.p` assigns the value io-param to a database field, then displays io-param.

r-runpr2.p

```
DEFINE VARIABLE io-param AS INTEGER NO-UNDO.

FOR EACH Item:
  DISPLAY Item.ItemName Item.OnHand WITH 1 DOWN.
  io-param = Item.OnHand.
  RUN r-param2.p (INPUT-OUTPUT io-param).
  Item.OnHand = io-param.
  DISPLAY io-param LABEL "New Quantity On-hand".
END.
```

r-param2.p

```
DEFINE INPUT-OUTPUT PARAMETER io-param AS INTEGER NO-UNDO.

DEFINE VARIABLE inp-qty AS INTEGER NO-UNDO.

PROMPT-FOR inp-qty LABEL "Quantity Received?".
ASSIGN inp-qty.
io-param = io-param + inp-qty.
```

The following example uses a buffer parameter. The procedure `r-bufp.p` passes the Customer buffer to the `getCustomer` internal procedure, which attempts to find a record using that buffer.

r-bufp.p

```
DEFINE BUTTON btnFind LABEL "Find Customer".

DO WITH FRAME frCustomer WITH SIDE-LABELS:
  ENABLE Customer.CustNum btnFind.

  ON CHOOSE OF btnFind DO:
    RUN getCustomer (Customer.CustNum:HANDLE, BUFFER Customer).
    IF NOT ERROR-STATUS:ERROR THEN
      DISPLAY Customer EXCEPT Customer.Comments WITH SIDE-LABELS.
  END.

  ON ENTRY OF Customer.CustNum
    HIDE MESSAGE.
END.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
PROCEDURE getCustomer:
  DEFINE INPUT PARAMETER  hWidget     AS HANDLE NO-UNDO.
  DEFINE PARAMETER BUFFER bufCustomer FOR Customer.

  FIND bufCustomer WHERE bufCustomer.CustNum =
    INTEGER(hWidget:SCREEN-VALUE) NO-LOCK NO-ERROR.
  IF NOT AVAILABLE bufCustomer THEN DO:
    MESSAGE "Customer record not found." VIEW-AS ALERT-BOX.
    RETURN ERROR.
  END.
END.
```

The following example defines parameters for the DLL routine, `MessageBox`, which displays a message on the screen:

r-dllex1.p

```
DEFINE VARIABLE result AS INTEGER NO-UNDO.

MESSAGE "  It's a whole new world!"
  VIEW-AS ALERT-BOX MESSAGE BUTTONS OK TITLE "ABL Message".

RUN MessageBoxA (0, "  It's a whole new world, again!!",
  "ABL DLL access", 0, OUTPUT result).

PROCEDURE MessageBoxA EXTERNAL "user32.dll":
  DEFINE INPUT PARAMETER hwnd    AS LONG.
  DEFINE INPUT PARAMETER mbtext  AS CHARACTER.
  DEFINE INPUT PARAMETER mbtitle AS CHARACTER.
  DEFINE INPUT PARAMETER style   AS LONG.
  DEFINE RETURN PARAMETER result AS LONG.
END.
```

Notes

- All procedure parameters are normally passed by value, by default. This means that for any INPUT-OUTPUT or OUTPUT parameter, the field or variable that receives the output value is not set by the called procedure until the procedure returns without error. An exception is made for local DATASET, DATASET-HANDLE, TABLE, and TABLE-HANDLE parameters, which you may pass by reference or by binding by specifying the parameter in a RUN statement using either the BY-REFERENCE or BIND option. If you specify the BIND option in the RUN statement, you must also specify the BIND option in the DEFINE PARAMETER statement.

For more information about passing parameters by reference or by binding, see the Parameter passing syntax reference entry.

- You cannot pass a BLOB or CLOB field as a parameter. To pass a BLOB or CLOB field as a parameter, you must include the field in a temp-table or convert the field to its MEMPTR or LONGCHAR counterpart, respectively.
- Buffer parameters are scoped in the same way as shared buffers. They also affect cursors defined in the calling procedure in the same way as shared buffers.
- For DLL or UNIX shared library routine declarations:
  - The LIKE *field* option can only specify a database field of type CHARACTER or a variable of type CHARACTER or MEMPTR.
  - The COLUMN, COLUMN-LABEL, DECIMALS, INITIAL, FORMAT, LABEL, and NO-UNDO options have no effect.
- You cannot pass a DATE, DATETIME, or DATETIME-TZ as a parameter to or from a DLL routine or a UNIX shared library routine.
- You can pass a LONGCHAR as a parameter to a DLL routine or a UNIX shared library routine. When passing a LONGCHAR parameter, the AVM passes only the text string (not the code page information). You are responsible for setting the code page of a LONGCHAR parameter.
- You can pass an array of type INTEGER, INT64, or DECIMAL as a parameter to or from a DLL routine or a UNIX shared library routine. You can pass an array of type CHARACTER to (not from) a DLL routine or a UNIX shared library routine.
- You cannot pass a variable or array that contains the Unknown value (?) to a DLL.
- RETURN parameters are supported only for DLL or UNIX shared library routines. The RETURN parameter type must match the OUTPUT parameter that returns the DLL function value in the RUN statement for the routine. You cannot pass an array as a RETURN parameter to DLL or UNIX shared library routines. Use a MEMPTR instead.
- If you specify a RETURN parameter as MEMPTR to return a character string, use the GET-STRING function to extract the CHARACTER value.
- For more information on DLL routine parameters and how they map to ABL data types, see the topics on DLLs in *OpenEdge Programming Interfaces*.
- You cannot pass a MEMPTR as a parameter to or from a COM object.
- You can pass a LONGCHAR, a DATETIME, a DATETIME-TZ, and an array as a parameter to or from a COM object.

  When passing a LONGCHAR parameter to a COM object, the AVM passes only the text string (not the code page information). When receiving a text string from a COM object into a LONGCHAR parameter, the AVM converts the text string to the code page associated with the LONGCHAR parameter only if the LONGCHAR has a fixed code page. Otherwise, the AVM sets the LONGCHAR code page to UTF-16. If the AVM cannot convert a LONGCHAR, it raises a run-time error.

  When passing a DATETIME or DATETIME-TZ parameter to a COM object, the AVM represents the time to the millisecond. When passing a DATETIME-TZ parameter to a COM object, the AVM first converts the DATETIME-TZ value relative to the local session's date and time, and then drops the time zone.

  When receiving a date from a COM object into a DATETIME or DATETIME-TZ parameter, the AVM represents the time to the millisecond. When receiving a date from a COM object into a DATETIME-TZ parameter, the AVM sets the time zone to the local session's time zone.
- For more information on ActiveX event parameters, or using COM objects, see *OpenEdge Programming Interfaces*.

- For dynamic temp-table parameters:
  - If the parameter is INPUT TABLE-HANDLE, the temp-table definition behind the handle plus the temp-table contents are sent from the caller to the called routine. The called routine may have either the dynamic INPUT TABLE-HANDLE or the compile-time defined INPUT TABLE as a matching parameter.
  - When running remotely with the application server, if the parameter is INPUT TABLE-HANDLE, a new instance of the temp-table is created along with its handle, completely separate from the caller's table, and is populated with the contents from the caller's table.
  - You can override this default behavior to allow the calling procedure and the called procedure to access the same object instance by passing the TABLE-HANDLE parameter by reference or by binding (that is, by specifying the parameter in a RUN statement using either the BY-REFERENCE or BIND option). If you specify the BIND option in the RUN statement, you must also specify the BIND option in the DEFINE PARAMETER statement.
  - The OUTPUT TABLE-HANDLE parameter, just like any other OUTPUT variable, cannot change the data in the caller procedure: in this case, the definition of the temp-table).
  - If the parameter is OUTPUT TABLE-HANDLE, the called routine sends back the definition behind the handle along with the contents of the output temp-table. In the caller, if the original handle was the Unknown value (?), a new instance of the temp-table is created and populated with the output contents. The temp-table object is created in the default widget-pool of the caller's OpenEdge session. To avoid memory leaks, the temp-table object must be manually destroyed. For more information on deleting objects, see the reference entry for the DELETE OBJECT statement.
  - If the original handle is not the Unknown value (?), the caller's existing table must match the table being received from the called routine.
  - If the APPEND option is used, the new data is added to the existing table's data.
  - If the parameter is INPUT-OUTPUT TABLE-HANDLE, a combination of the above occurs.
  - If you call a remote procedure asynchronously (using the ASYNCHRONOUS option of the RUN statement) and pass a parameter as OUTPUT TABLE-HANDLE *temp-table-handle* APPEND, the event procedure or method must specify a corresponding DEFINE INPUT PARAMETER TABLE-HANDLE FOR *temp-table-handle* APPEND statement, and *temp-table-handle* must be global to both the calling procedure and the event procedure or method.
- If you define an INPUT TABLE parameter for an asynchronous event procedure with a data type that is different from the data type of the corresponding OUTPUT TABLE parameter passed from the application server, any failure to convert the passed value causes the event procedure or method to fail and the AVM to display an error message on the client.
- The AVM creates the temp-table object for an OUTPUT TABLE-HANDLE parameter in the session's default widget pool. Remember to delete this object at the appropriate point to avoid memory leaks.
- You cannot specify a ProDataSet object or ProDataSet object handle as a parameter for an asynchronous remote procedure.
- For dynamic ProDataSet object parameters:
  - If the parameter is INPUT DATASET-HANDLE, the ProDataSet object definition behind the handle plus the ProDataSet object contents are sent from the caller to the called routine. The called routine may have either the dynamic INPUT DATASET-HANDLE or the compile-time defined INPUT DATASET as a matching parameter.
  - If the parameter is INPUT DATASET-HANDLE, a new instance of the ProDataSet object is created along with its handle, completely separate from the caller's table, and is populated with the contents from the caller's table.

- You can override this default behavior to allow the calling procedure and the called procedure to access the same object instance by passing the DATASET-HANDLE parameter by reference or by binding (that is, by specifying the parameter in a RUN statement using either the BY-REFERENCE or BIND option). If you specify the BIND option in the RUN statement, you must also specify the BIND option in the DEFINE PARAMETER statement.
- If the parameter is OUTPUT DATASET-HANDLE, the handle plus the definition behind the handle are sent from the caller to the called routine. The called routine may have either the dynamic OUTPUT DATASET-HANDLE or the compile-time defined OUTPUT DATASET as a matching parameter.
- If the parameter is OUTPUT DATASET-HANDLE, and the caller's handle is the Unknown value (?), no definition is sent to the called routine.
- If the parameter is OUTPUT DATASET-HANDLE, the called routine sends back the definition behind the handle along with the contents of the output ProDataSet object. In the caller, if the original handle is the Unknown value (?), a new instance of the ProDataSet object is created and populated with the output contents. If the original handle is not the Unknown value (?), the caller's existing object must match the object being received from the called routine.
- If the APPEND option is used, the new data is added to the existing object's data.
- If the parameter is INPUT-OUTPUT DATASET-HANDLE, a combination of the above occurs.

- A BUFFER parameter cannot be a REFERENCE-ONLY buffer or a BEFORE-TABLE buffer. The use of them as a BUFFER type of parameter (as opposed to INPUT or OUTPUT) is a compiler error. Instead of passing the buffer, the buffer's table can be passed by-reference.

See also

DEFINE BUFFER statement, DEFINE VARIABLE statement, DELETE PROCEDURE statement, Parameter passing syntax, RUN statement, Type-name syntax, USING statement, VAR statement

DEFINE PROPERTY statement

Note: This statement is applicable only when used in a class or interface definition (`.cls`) file.

Syntax

Use the following syntax for defining a property that you can both read and write:

```
DEFINE [ PRIVATE | PACKAGE-PRIVATE | PROTECTED | PACKAGE-PROTECTED | PUBLIC ]
       [ STATIC | ABSTRACT ] [ OVERRIDE ][ SERIALIZABLE | NON-SERIALIZABLE ]
  PROPERTY property-name
  {{
       AS primitive-type-name |
       AS [ CLASS ]object-type-name
    } [ EXTENT [constant ] ]}
  [ INITIAL { constant  | {[constant[ , constant]...]}}]
  [ SERIALIZE-NAME serialize-name ][ NO-UNDO ]
  {
       [accessor-access-mode] GET [implementation] .
       SET [implementation] .
    |
       GET [implementation] .
       [accessor-access-mode] SET [implementation] .
  }
```

Use the following syntax to define a property that you can read only:

```
DEFINE [ PRIVATE | PACKAGE-PRIVATE | PROTECTED | PACKAGE-PROTECTED | PUBLIC ]
       [ STATIC | ABSTRACT ] [ OVERRIDE ][ NON-SERIALIZABLE ]
  PROPERTY property-name
  {{
       AS primitive-type-name |
       AS [ CLASS ]object-type-name
    }[ EXTENT [constant ]]}
  [ INITIAL { constant |{[constant[ , constant]...]}}]
  [ SERIALIZE-NAME serialize-name ][ NO-UNDO ]

  GET [implementation] .
```

Use the following syntax to define a property that you can write only:

```
DEFINE [ PRIVATE | PACKAGE-PRIVATE | PROTECTED | PACKAGE-PROTECTED | PUBLIC ]
       [ STATIC | ABSTRACT ] [ OVERRIDE ][ NON-SERIALIZABLE  ]
  PROPERTY property-name
  {{
       AS primitive-type-name |
       AS [ CLASS ]object-type-name
    }[ EXTENT [constant ]]}
  [ INITIAL { constant |{[[ , constant].. .]}}]
  [ SERIALIZE-NAME serialize-name ][ NO-UNDO ]

  SET [implementation] .
```

Use the following syntax to declare an interface property prototype:

```
DEFINE [ PUBLIC ] PROPERTY property-name
  {{ AS primitive-type-name | AS [ CLASS ]object-type-name}
     [ EXTENT [constant ]][ SERIALIZE-NAME serialize-name ]
     [ NO-UNDO ]}
  { GET.
    SET.
    |
    GET.
    |
    SET. }
```

Use the following syntax to declare an abstract property prototype:

```
DEFINE [ PACKAGE-PRIVATE | PROTECTED | PACKAGE-PROTECTED | PUBLIC ]
  ABSTRACT PROPERTY property-name
  {{ AS primitive-type-name | AS [ CLASS ]object-type-name}
     [ EXTENT [constant ]][ SERIALIZE-NAME serialize-name ]
     [ NO-UNDO ]}
  { GET.
    SET.
    |
    GET.
    |
    SET. }
```

[PRIVATE | PACKAGE-PRIVATE | PROTECTED | PACKAGE-PROTECTED | PUBLIC]

Specifies the access mode for the property. A PRIVATE property can be accessed by the defining class. An instance can access a private property of another instance if both instances are of the same class. A PACKAGE-PRIVATE property can be accessed from within the class and any class within its package. A PROTECTED property can be accessed by the defining class and any of its derived classes. A PACKAGE-PROTECTED property can be accessed from within the class, any class within its package, and from within any subclass that inherits the class. A PUBLIC property can be accessed from any code (in a class or a procedure) that has access to a class instance.

The default access mode is PUBLIC.

This access mode applies to both reading and writing the property value by default. However, you can set a different access mode for reading or writing the property value (but not both) using an appropriate *accessor-access-mode* option to define the corresponding accessor (GET or SET).

When declaring an interface property prototype, the access mode for the property must be PUBLIC (the default).

When defining an abstract property, the access mode for the property cannot be PRIVATE.

[STATIC]

Defines a property that is a static member of the class type for which it is defined and that is scoped to the ABL session where it is referenced. ABL creates one copy of the specified class static property on first reference to the class type, and ABL creates only one such copy for any number of instances of the class that you create. You can reference an accessible static property in any piece of code.

Without this option, ABL defines an instance property that is scoped to a single instance of the class where it is defined. ABL creates one copy of the specified instance property for each such class instance that you create. You can reference any public instance property (non-abstract) in any procedure, or in any instance or static method defined inside or outside of the class where the instance property is defined. Any static method can reference the public instance property only using an object reference to a class instance that defines the property as a member. If the referencing static method is defined in the same class as the public instance property, the class must instantiate itself in order to have access to an instance reference.

For more information on the mechanism for accessing properties of different access modes and scopes, see the reference entry for Class-based property access.

The STATIC option is not valid when you:

- Define or implement an abstract property
- Declare an interface property prototype
- Implement an interface property

Note: The access mode modifier and the STATIC/ABSTRACT modifier can appear in either order, e.g., `DEFINE STATIC PUBLIC myProperty PROPERTY...` is valid syntax.

[ABSTRACT]

Defines the property as an abstract member of the class type for which it is defined. The defining class type must also be abstract. If you define an abstract property, it has the following requirements:

- You must specify the OVERRIDE option if an inherited abstract property has the same name and data type.
- You can specify any access mode other than PRIVATE.

- The abstract property must be overridden and implemented in a derived class.

Any class definition for an instance property that includes the OVERRIDE option and does not include the ABSTRACT option defines a property implementation. If it also includes the ABSTRACT option, the inherited abstract property remains abstract.

This option is not valid either when you define a static property or when you declare an interface property prototype.

[OVERRIDE]

[SERIALIZABLE | NON-SERIALIZABLE]

Class-based objects that are defined as serializable (using the SERIALIZABLE option in the CLASS statement) can be passed as parameters in remotes call between the application server and ABL clients and can be serialized to binary or JSON format. By default, both passing a class as a parameter and serializing an object to binary via the Progress.IO.BinarySerializer class include all data members regardless of access mode. However, for JSON serialization via Progress.IO.JsonSerializer, only public data members are serialized. To include a non-public property during JSON serialization, SERIALIZABLE must be added to the definition. See Serialize( ) method (JsonSerializer) for more information.

Use the NON-SERIALIZABLE option to exclude a given property from parameter passing between an application server and ABL client and from the serialization process via the Progress.IO.BinarySerializer or Progress.IO.JsonSerializer class. A property marked as NON-SERIALIZABLE is assigned its initial value when the class is deserialized.

You can also use the NON-SERIALIZABLE option to make a class that otherwise would not be serializable because one or more of its members. (See the Parameter passing syntax entry for full details on what can and cannot be serialized.) For example, if a property itself is a class-based object that is not serializable, adding the NON-SERIALIZABLE keyword to that property will allow the class that contains it to be defined as SERIALIZABLE.

Note: Use the SERIALIZABLE option only with properties that are both readable and writable. This option is not for use with properties defined as STATIC, as ABSTRACT, or as part of an interface.

---

property-name

Specifies the name of the property. The *property-name* must be unique among the names of all properties, events, and variable data members that are defined in the class and its inherited class hierarchy (not counting abstract member overrides) and that are accessible to the defining class (not defined as PRIVATE in a super class).

Note: Members of a class are grouped into six namespaces, including buffers/temp-tables, methods, variables/properties/events, ProDataSets, queries, and data-sources. Variables, properties, and events defined as members of a class share the same namespace. There can be only one class member in this namespace with a given name (not counting abstract member overrides).

You can reference this name from outside the property definition in order to read a value from or write a value to the property. You can reference this name from inside the property definition (in a GET or SET accessor) in order to read or write the value of the default memory for the property, according to its data type.

Note: If the property is defined in a class that is derived from a .NET class, other restrictions on *property-name* apply. For more information, see the CLASS statement reference entry.

AS *primitive-type-name*

Specifies a built-in primitive type (*primitive-type-name*) and an optional initial value (*constant*) for the default memory of the property. The built-in data type can be one of the following:

```
CHARACTER | COM-HANDLE | DATE | DATETIME | DATETIME-TZ
  | DECIMAL | HANDLE | INT64 | INTEGER | LOGICAL | LONGCHAR
  | MEMPTR | RAW | RECID | ROWID |  AS-data-type
```

If you are overriding a .NET abstract property or implementing a property defined in a .NET interface, *primitive-type-name* must specify the exact same .NET mapped data type that is defined for the corresponding .NET property. For a .NET mapped data type that is a default match for a given ABL primitive type, you must use the default matching ABL data type, as shown in Table 24: Implicit mappings between .NET and ABL data types. (For example, INTEGER indicates a .NET System.Int32.) For a .NET mapped data type that is **not** a default match for one of the ABL primitive types, ABL provides a data type keyword (*AS-data-type*) that you must use to explicitly indicate the required .NET data type, as shown in Table 25: Explicit mappings between ABL and .NET data types. (For example, the AS data type, UNSIGNED-BYTE, indicates a .NET System.Byte.)

Note: At run time, a property defined using an *AS-data-type* keyword behaves in ABL like the corresponding ABL primitive type shown in Table 25: Explicit mappings between ABL and .NET data types. (For example, an UNSIGNED-BYTE behaves like an INTEGER.)

Also note that when implementing a .NET array property, you must specify the .NET array object type (for example, "System.Int32[]" or "System.Byte[]"); you **cannot** use an ABL array equivalent (such as INTEGER EXTENT or UNSIGNED-BYTE EXTENT).

These are the same set of primitive types that can be specified for the return type of a method. Thus, for example, a property cannot have a complex type such as a temp-table, ProDataSet, or any other data type that is invalid as a method return type. For more information on these primitive types, see the Data types reference entry.

Note: To manage public access for a complex data type that cannot be specified for a property, you can define both a data member of that complex type and a separate public method that passes the complex type as a parameter.

AS [CLASS] *object-type-name*

Specifies an object reference with the data type of a class or interface for the default memory of the property. The default value of the property is the Unknown value (?). You cannot assign an initial value using the INITIAL option.

object-type-name

Specifies the type name of an ABL or .NET class or interface. Specify an object type name using the syntax described in the Type-name syntax reference entry. With an appropriate USING statement, you can also specify a class or interface name alone, without the qualifying package or namespace.

You cannot directly specify the type name of a .NET mapped object type (such as System.Int32). To define a property that matches a .NET mapped type, you must define it as the corresponding ABL primitive type (*primitive-type-name*).

If you are overriding a .NET abstract property or implementing a property defined in a .NET interface, *object-type-name* must specify the exact same .NET object type as the corresponding interface property. However, for .NET inner (nested) type, note the difference in the ABL syntax, which replaces the corresponding period (.) in the .NET object type with a plus (+) (see the Type-name syntax reference entry).

Also note that when implementing a .NET array property, you must specify the exact .NET array object type (for example, `"System.Drawing.Point[]"`); you **cannot** use an ABL array equivalent (such as System.Drawing.Point EXTENT).

`CLASS`

If the specified class or interface type name conflicts with an abbreviation of a built-in primitive type name, such as INT for INTEGER, you must specify the CLASS keyword.

For more information on object references, see the Class-based object reference reference entry.

[`EXTENT` **[** *constant* **]]**

Defines the property as an array of data elements, where the element data type is specified by either the AS *primitive-type-name* option or the AS *object-type-name* option. This option can specify an array property as either determinate (has a defined number of elements) or indeterminate (has an undefined number of elements). To define a determinate array property, specify the `EXTENT` option with the *constant* argument. This optional argument is an integer value that represents the number of elements in the property array. To define an indeterminate array property, specify the `EXTENT` option without the *constant* argument.

The `EXTENT` is part of the property data type. For more information, see the Type-name syntax reference entry.

For more information about arrays, see Work with one-dimensional arrays in *Basic ABL*.

If you do not use the `EXTENT` option (or you specify *constant* as 0), the property is not an array property.

[`INITIAL` **{***constant* **|** `[` *constant* **[** `,` *constant* **]** `. . . ]` **}]**

The initial value of the property you want to define. If you use the AS *primitive-type-name* option and you do not use the `INITIAL` *constant* option, the initial value is the default initial value for the data type of the variable or array element.

When you define an array property, you can supply initial values for each element in the array. For example:

```
DEFINE PUBLIC PROPERTY pbarray-var AS AS CHARACTER EXTENT 3 INITIAL
["Add","Delete","Update"]
      GET.
      SET.
```

If you do not supply enough values to fill up the elements of the array, the AVM puts the last value you named into the remaining elements of the array. If you supply too many values, the AVM raises an error.

If you define a property as an indeterminate array, and you supply initial values for elements in the array, the AVM sets the number of elements and treats the array as a determinate array.

The `INITIAL` option is not valid either when you define an abstract property or when you declare an interface property prototype.

The following table lists the default initial values for the various property data types.

Table 33: Default variable initial values

Data type	Default initial value
CHARACTER	"" (an empty string)
CLASS*, **	Unknown value (?)
COM-HANDLE**	Unknown value (?)
DATE	Unknown value (?) (displays as blanks)
DATETIME	Unknown value (?)
DATETIME-TZ	Unknown value (?)
DECIMAL	0
HANDLE**	Unknown value (?)
INT64	0
INTEGER	0
LOGICAL	no
LONGCHAR	Unknown value (?)
MEMPTR**	A zero-length sequence of bytes
RAW**	A zero-length sequence of bytes
RECID	Unknown value (?)
ROWID**	Unknown value (?)

* If you display a class instance using the MESSAGE statement, ABL automatically invokes the ToString( ) method (Object) (provided by the Progress.Lang.Object class) on the object reference. To display a class instance in a frame (for example, using the DISPLAY statement), you must first explicitly convert the object reference to a displayable type using the INT64 function, the INTEGER function, the STRING function, or the ToString( ) method (Object).

** You cannot use the INITIAL option to specify an initial value for this data type as part of the property definition.

SERIALIZE-NAME *serialize-name*

An optional CHARACTER constant that specifies the name of the property as it should appear when serialized into JSON. The default is *property-name*. Use this option when the serialized name contains invalid characters for an ABL name, is an ABL keyword, or an invalid JSON property name.

This option cannot be used if the property is defined as NON-SERIALIZABLE or the class that contains it is not defined as SERIALIZABLE.

[NO-UNDO **]**

When the value of a property is changed during a transaction and the transaction is undone, the AVM restores the value of the property to its prior value. If you do not want, or if you do not need, the value of such a property to be undone even when it has been changed during a transaction, use the NO-UNDO option. NO-UNDO properties are more efficient; use this option whenever possible.

[*accessor-access-mode* **]**

Specifies the access mode for the immediately following accessor (GET or SET). The possible access modes include:

PACKAGE-PROTECTED | PROTECTED | PACKAGE-PRIVATE | PRIVATE

To use an *accessor-access-mode* with accessors:

- You must define at least one accessor (GET or SET) that defaults to the access mode of the property. Therefore, if you define only one accessor, you cannot specify a separate *accessor-access-mode* for it.
- If you define two accessors, you can specify an *accessor-access-mode* for one of the accessors, but only if this *accessor-access-mode* is more restrictive than the access mode of the property.
- If you are defining an abstract property, you cannot specify the PRIVATE access mode for an accessor.

For example, if the property access mode is PUBLIC, any *accessor-access-mode* that you specify must be PACKAGE-PROTECTED, PROTECTED, PACKAGE-PRIVATE, or PRIVATE; if the property access mode is PROTECTED, any accessor-access-mode that you specify must be PRIVATE, unless the property is abstract.

GET **[** *implementation* **]** .

An accessor (GET) that defines the property as readable. If you do not define a GET accessor for a class property, the property cannot be read, not even from within the defining class.

It is valid to omit the GET accessor when defining a property for an interface. It is then up to the class implementing the interface to decide whether to define the GET accessor for the property or not. However, as stated above, if you do not define a GET accessor for a class property, the property cannot be read. If you do not define a GET accessor, then you **must** define a SET accessor.

You can define a GET accessor using two basic forms: with or without an *implementation*. However, if you are defining an abstract property or an interface property prototype, you cannot define the GET accessor with an implementation. This is the syntax for a GET accessor **with** an implementation:

```
GET ( [array-index-parameter] ) : get-logic END [ GET ] .
```

array-index-parameter

Specifies an INPUT parameter that provides access to the index value of the current element of an array property from within the *get-logic*. When a property is defined as an array using the EXTENT option, the GET accessor references the single element of the property array. The *array-index-parameter* specifies the subscript value for the element being accessed. This is the syntax:

```
[ INPUT ] array-index-name AS { INTEGER | INT64 }
```

array-index-name

Specifies the name of the INPUT parameter that provides the subscript value (index) for this array element. You must use this value in a subscript on the property name (*property-name*[*array-index-name*]) to access the default memory for the current array element.

INTEGER | INT64

Specifies the data type of the index. Use the data type that supports the largest dimension that is defined for or that you expect to be fixed for the property array at run time.

Where supported, an unsubscripted array reference is a reference to the whole array. For a property array, an unsubscripted read reference invokes the GET accessor for each array element in ascending order by subscript value.

get-logic

Can contain ABL code that executes for any operation that reads the property from outside its own GET accessor definition. This code can include any ABL statements that are valid in a method of a class, including CATCH and FINALLY blocks. However, if you define the property itself as static, you cannot access any instance members of classes (including the defining class) or use the SUPER and THIS-OBJECT system references; you can only access static class members and the local data elements of the GET accessor. The actual value that you return from the property is entirely dependent on your *get-logic* code. Within *get-logic*, any operation that reads from *property-name* (such as the right-hand side of an assignment) directly reads the value of the property's default memory. However, any operation that writes to *property-name* within *get-logic* invokes the property's own SET accessor to assign the value, exactly like writing the property from outside of its own definition. If the SET accessor is defined with an implementation, that implementation determines how the value is written to the property.

You do not have to use the property's default memory to provide the value read from a property. You can also use any other accessible data, such as a data member, as storage to access property values. To return a value from *get-logic* to any outside reader of the property, you must return a value with the correct data type using the RETURN statement (similar to any method that returns a value). If you do not invoke the RETURN statement in *get-logic*, the property returns the Unknown value (?).

This is the syntax for a GET accessor **without** an implementation:

```
GET.
```

Without an implementation, an operation that reads the property directly accesses the current value in the property's default memory.

If the property is an array, the default accessor implementation handles the subscript references automatically. Providing an *array-index-parameter* in this case is a compiler error.

```
SET [ implementation ] .
```

An accessor (SET) that defines the property as writable. If you do not define a SET accessor, the property cannot be written, not even from within the defining class. You can define a SET accessor using two basic forms: with or without an implementation.

It is valid to omit the SET accessor when defining a property for an interface. It is then up to the class implementing the interface to decide whether to define the SET accessor for the property or not. However, as stated above, if you do not define a SET accessor for a class property, the property cannot be written. If you do not define a SET accessor, then you **must** define a GET accessor.

You can define a SET accessor using two basic forms: with or without an *implementation*. However, if you are defining an abstract property or an interface property prototype, you cannot define the SET accessor with an implementation. This is the syntax for a SET accessor **with** an implementation:

```
SET ( parameter-definition [ , array-index-parameter ] ) :
  set-logic END [ SET ] .
```

parameter-definition

Specifies an INPUT parameter that provides access to the value written to the property from within the *set-logic*. The *set-logic* can contain ABL code that executes for any operation that writes to the property from outside its own SET accessor definition. This code can include any ABL statements that are valid in a method of a class, including CATCH and FINALLY blocks. However, if you define the property itself as static, you cannot access any instance members of classes (including the defining class) or use the SUPER and THIS-OBJECT system references; you can only access static class members and the local data elements of the SET accessor. To access the value being written to the property, specify *parameter-definition* using the following syntax:

```
[ INPUT ] parameter-name AS
  {primitive-type-name | [ CLASS ] object-type-name}
```

[INPUT] *parameter-name*

Specifies the name of the INPUT parameter that you reference in *set-logic* to obtain the value written to the property.

AS **{***primitive-type-name* **| [** CLASS **]** *object-type-name***}**

Specifies the data type of the parameter as a built-in primitive type (*primitive-type-name*), or as a built-in or user-defined object type (*object-type-name*). You must specify exactly the same data type as you have defined for the property.

array-index-parameter

Specifies an INPUT parameter that provides access to the index value of the current element of an array property from within the *set-logic*. When a property is defined as an array using the EXTENT option, the SET accessor references a single element of the property array. The *array-index-parameter* specifies the subscript value for the element being accessed. This is the syntax:

```
[ INPUT ] array-index-name AS { INTEGER | INT64 }
```

array-index-name

Specifies the name of the INPUT parameter that provides the subscript value (index) for this array element. You must use this value in a subscript on the property name (*property-name*[*array-index-name*]) to access the default memory for the current array element.

INTEGER **|** INT64

Specifies the data type of the index. Use the data type that supports the largest dimension that is defined for or that you expect to be fixed for the property array at run time.

Where supported, an unsubscripted array reference is a reference to the whole array. For a property array, an unsubscripted write reference invokes the SET accessor for each array element in ascending order by subscript value.

set-logic

Can contain ABL code that accesses the value written to the property as *parameter-name*, then uses *parameter-name* (if you choose) to set the new value for the property. The actual value that you use to set the property is entirely dependent on your *set-logic* code. Within *set-logic*, any operation that writes to *property-name* (such as the left-hand side of an assignment) directly writes the specified value to the property's default memory. However, any operation that reads from *property-name* within *set-logic* invokes the property's own GET accessor to read the value, exactly like reading the property from outside of its own definition. If the GET accessor is defined with an implementation, that implementation determines the value that is read from the property.

You do not have to use the property's default memory to store the property value. You can use any other accessible data, such as a data member, as storage to store property values. If you do not write a value to *property-name* within *set-logic*, the property's default memory retains whatever value it had prior to any operation that writes to the property.

This is the syntax for a SET accessor **without** an implementation:

```
SET.
```

Without an implementation, any operation that writes the property writes the value directly to the property's default memory without passing through *parameter-name*.

If the property is an array, the default accessor implementation handles the subscript references automatically. Providing an *array-index-parameter* in this case is a compiler error.

Note: This is equivalent to, but more efficient than, using an implementation to explicitly assign *property-name* the value of *parameter-name*.

Examples

The examples that follow show two different ways to access the same private data of a class using properties.

The first example, shows a class (r-DefineProperties1) that defines a PUBLIC property (cCurrentSalesRepName) followed by a procedure that accesses this property. The property is defined with two accessors without implementations, providing direct access to the property value. The GET accessor is PUBLIC, but the SET accessor is PRIVATE, allowing only the class or an instance of the class to set the property value. In this case, the class sets the property from data (RepName field) in the Sports2020 database provided by a buffer (bSalesRep), which is PRIVATE.

The class also provides a PUBLIC method (getNextSalesRep( )) to read the SalesRep table one record at a time and set the cCurrentSalesRepName property to the value of the RepName field for each record. This PUBLIC method also uses a PRIVATE method (restartSalesRep( )) to reset the record position to the beginning of the table, based on an INPUT parameter. The class constructor also uses this PRIVATE method to initialize the record buffer to the first record in the table. The class sets the cCurrentSalesRepName property to the Unknown value (?) if the SalesRep table is empty or getNextSalesRep( ) reaches the end of the SalesRep table.

r-DefineProperties1.cls

```
CLASS r-DefineProperties1:

  DEFINE PUBLIC PROPERTY cCurrentSalesRepName AS CHARACTER NO-UNDO
    GET.
    PRIVATE SET.
  DEFINE PRIVATE BUFFER bSalesRep FOR SalesRep.

  CONSTRUCTOR PUBLIC r-DefineProperties1 ():
    restartSalesRep() NO-ERROR.
    IF ERROR-STATUS:ERROR THEN RETURN ERROR ERROR-STATUS:GET-MESSAGE(1).
  END CONSTRUCTOR.

  METHOD PRIVATE VOID restartSalesRep ():
    FIND FIRST bSalesRep NO-ERROR.
    IF NOT AVAILABLE bSalesRep THEN DO:
      cCurrentSalesRepName = ?.
      RETURN ERROR "SalesRep table empty".
    END.
    ELSE
      cCurrentSalesRepName = bSalesRep.RepName.
  END METHOD.

  METHOD PUBLIC VOID getNextSalesRep (INPUT lRestart AS LOGICAL):
    IF lRestart THEN DO:
      restartSalesRep() NO-ERROR.
      IF NOT AVAILABLE bSalesRep THEN
        RETURN ERROR ERROR-STATUS:GET-MESSAGE (1).
    END.
    ELSE DO:
      FIND NEXT bSalesRep NO-ERROR.
      IF NOT AVAILABLE bSalesRep THEN
        cCurrentSalesRepName = ?.
      ELSE
        cCurrentSalesRepName = bSalesRep.RepName.
    END.
  END METHOD.

END CLASS.
```

The following procedure (`r-runDefineProperties1.p`) instantiates the `r-DefineProperties1` class, referenced by `clProps`, and reads and displays the value of the `clProps:cCurrentSalesRepName` property in a message, starting with the first `SalesRep` record found as part of class instantiation. The procedure then displays the value of `clProps:cCurrentSalesRepName` in a message for each record found by the `clProps:getNextSalesRep( )` method, restarting from the beginning of the `SalesRep` table at the direction of the user.

r-runDefineProperties1.p

```
DEFINE VARIABLE clProps AS CLASS r-DefineProperties1 NO-UNDO.

clProps = NEW r-DefineProperties1() NO-ERROR.
IF ERROR-STATUS:ERROR THEN DO:
  MESSAGE ERROR-STATUS:GET-MESSAGE(1) VIEW-AS ALERT-BOX.
  QUIT.
END.

DO WHILE NOT clProps:cCurrentSalesRepName = ?:
  MESSAGE"The current sales rep is "
    clProps:cCurrentSalesRepName + ".  Continue?"
    VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO UPDATE lContinue AS LOGICAL.
  IF NOT lContinue THEN LEAVE.
  ELSE DO:
    clProps:getNextSalesRep(FALSE).
    IF clProps:cCurrentSalesRepName = ? THEN DO:
      MESSAGE "End of sales rep list.  Restart?"
        VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO
        UPDATE lRestart AS LOGICAL.
      IF lRestart THEN
        clProps:getNextSalesRep(TRUE).
    END.
  END.
END.
```

In the next example, the `r-DefineProperties2` class defines a `PUBLIC` property (`cNextSalesRepName`) that provides the same data as the `cCurrentSalesRepName` property defined by the `r-DefineProperties1` class. However, the `GET` accessor of the `cNextSalesRepName` property is also used to provide the same access to the `SalesRep` table that the `getNextSalesRep( )` method provides for the `r-DefineProperties1` class. So, the value of `cNextSalesRepName` changes with each access.

r-DefineProperties2.cls

```
CLASS r-DefineProperties2:

  DEFINE PUBLIC PROPERTY lSalesRepRestart AS LOGICAL NO-UNDO
    PRIVATE GET.
    SET.
  DEFINE PRIVATE BUFFER bSalesRep FOR SalesRep.

  DEFINE PUBLIC PROPERTY cNextSalesRepName AS CHARACTER
    GET ():
      IF lSalesRepRestart THEN DO:
        restartSalesRep() NO-ERROR.
        IF NOT AVAILABLE(bSalesRep) THEN
          RETURN ERROR ERROR-STATUS:GET-MESSAGE(1).
      END.
      ELSE DO:
        FIND NEXT bSalesRep NO-ERROR.
        IF NOT AVAILABLE(bSalesRep) THEN
          cNextSalesRepName = ?.
        ELSE
          cNextSalesRepName = bSalesRep.RepName.
      END.
      RETURN cNextSalesRepName.
    END GET.
    PRIVATE SET.

  CONSTRUCTOR PUBLIC r-DefineProperties2
    (OUTPUT cFirstSalesRepName AS CHARACTER):
    ASSIGN
      lSalesRepRestart   = TRUE
      cFirstSalesRepName = cNextSalesRepName.
  END CONSTRUCTOR.

  METHOD PRIVATE VOID restartSalesRep ():
    lSalesRepRestart = FALSE.
    FIND FIRST bSalesRep NO-ERROR.
    IF NOT AVAILABLE bSalesRep THEN DO:
      cNextSalesRepName = ?.
      RETURN ERROR "SalesRep table is empty".
    END.
    ELSE
      cNextSalesRepName = bSalesRep.RepName.
  END METHOD.

END CLASS.
```

Because this `cNextSalesRepName` property incorporates the record access provided by the `getNextSalesRep( )` method, the following procedure that accesses the `cNextSalesRepName` property must also use the property in a manner similar to how the `r-runDefineProperties1.p` procedure uses the `getNextSalesRep( )` method. As a result, the class also provides a second, publicly writable property (`lSalesRepRestart`) to indicate when the reading of `SalesRep` records must restart from the beginning of the table. (Note that the `getNextSalesRep( )` method provides its own `INPUT` parameter to indicate whether to restart record reading.)

Finally, the `r-DefineProperties2` class constructor sets `lSalesRepRestart` together with an initial read of the `cNextSalesRepName` property in order to initialize the record buffer to the first record of the table and pass the associated `RepName` field value to the instantiating procedure as an `OUTPUT` parameter.

Thus, the following procedure (r-runDefineProperties2.p) instantiates the r-DefineProperties2 class, referenced by clProps, and reads and displays the data from the clProps:cNextSalesRepName property in a manner similar to how the r-runDefineProperties1.p procedure reads and displays the same data using the getNextSalesRep( ) method. However, because the clProps:cNextSalesRepName property always returns the RepName field for the **next** SalesRep record in the table, the r-runDefineProperties2.p procedure must provide a separate variable (cCurrentSalesRepName) of its own, which provides the same function that the cCurrentSalesRepName property provides for the r-DefineProperties1 class, which is to maintain a current value read from the RepName field.

r-runDefineProperties2.p

```
DEFINE VARIABLE clProps             AS CLASS r-DefineProperties2 NO-UNDO.
DEFINE VARIABLE cCurrentSalesRepName AS CHARACTER NO-UNDO.

clProps = NEW r-DefineProperties2(OUTPUT cCurrentSalesRepName) NO-ERROR.

DO WHILE NOT cCurrentSalesRepName = ?:
  MESSAGE "The current sales rep is " cCurrentSalesRepName + ".  Continue?"
    VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO
    UPDATE lContinue AS LOGICAL.
  IF NOT lContinue THEN LEAVE.
  ELSE DO:
    cCurrentSalesRepName = clProps:cNextSalesRepName.
    IF cCurrentSalesRepName = ? THEN DO:
      MESSAGE "End of sales rep list.  Restart?"
        VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO
        UPDATE lRestart AS LOGICAL.
      IF lRestart THEN
        ASSIGN
          clProps:lSalesRepRestart = TRUE
          cCurrentSalesRepName     = clProps:cNextSalesRepName.
    END.
  END.
END.
```

Thus, the logic of the r-runDefineProperties2.p procedure is almost identical to the r-runDefineProperties1.p procedure, reading a clProps:cNextSalesRepName property instead of invoking a clProps:getNextSalesRep( ) method in order to read an appropriate value from the SalesRep table.

For more examples of property definitions, including static and abstract properties, see the descriptions of r-CustObj.cls, r-CustObjStatic.cls, and r-CustObjAbstract.cls in the CLASS statement reference entry.

Notes

- You cannot define a property within a procedure (internal or external) or within a method of a class (constructor or destructor). It can only be defined as a member of a class or as a prototype in an interface. For more information about declaring property prototypes in an interface, see the INTERFACE statement reference entry.
- PUBLIC, PACKAGE-PROTECTED, PROTECTED, and PACKAGE-PRIVATE properties defined within a class definition (.cls) file maintain their characteristics throughout the inherited class hierarchy. Thus, you cannot override properties in a subclass that are defined in a super class.
- You must define at least one GET or SET accessor, or define both accessors, for the property to be valid.
- You never invoke the GET and SET accessors defined for a property directly. These methods are only invoked implicitly when you read (GET) or write (SET) the value of the property. For example, you can read or write the property by using the property in an Assignment (=) statement or by passing the property as a parameter to a method, procedure, or user-defined function. For information on reading and writing property

values, see the Class-based property access reference entry. As an alternative, you can define a method in a class to read or write the value of a data member that you separately define in the same class.

- If the property name is an ABL reserved keyword, you might need to use the THIS-OBJECT system reference or Type-name syntax to reference the property. For more information on referencing properties, see the reference entry for Class-based property access.
- To avoid any unexpected performance impact when using the property, ensure that the body of any GET or SET accessor minimizes processing that might incur delay when referencing the property.
- You cannot specify formatting information in a property definition. You can control property formatting using the same default rules for formatting a variable or data member:

  1. Rely on the default formatting for the data type of the property. For more information, see the Data types reference entry.
  2. Add formatting information when you access the property in a statement (for example, using a DISPLAY statement).

- You can handle application errors in an a property accessor as in any ABL block. By executing a RETURN ERROR action at the block level or a THROW action at the block level with the presence of a ROUTINE-LEVEL ON ERROR UNDO, THROW statement, the AVM returns the ERROR condition to the statement that references the property and works much like an error raised by a method. If a RETURN ERROR also includes the option to return a character string value, or you set the ReturnValue property of a Progess.Lang.AppError object that you THROW, you can get this value using the RETURN-VALUE function following the statement that references the property or in a CATCH block that catches the Progress.Lang.AppError object. If the body of an accessor contains an UNDO block, any unhandled ERROR condition in that block undoes only the data within that block, according to the NO-UNDO setting of the data. The property value, itself, is not undone unless the property is defined without NO-UNDO. For more information, see *Develop Object-oriented ABL Applications*.
- If ERROR is raised during execution of an Assignment (=) statement, the value on the left-hand side usually remains unchanged from its value prior to the assignment. However, if the left-hand side of the assignment is a property and its SET accessor invokes the RETURN ERROR statement (raising ERROR on the assignment), the value of the property can be changed. This is because the SET accessor is a method, and like all methods that raise ERROR, any data elements that the SET accessor changes retain their most recent values after ERROR is raised. Thus, if the SET accessor changes the property value before invoking RETURN ERROR, the property retains its most recent change in value after ERROR is raised on the assignment.
- If an ABL property that implements a property defined in a .NET interface is accessed from .NET and it raises ERROR out of the accessor block, ABL returns a .NET System.Exception to the caller. If the error is raised by executing a RETURN ERROR with the optional error string, the Message property of the System.Exception describes the operation where the error occurred, but the error string is available only to the ABL session, using the RETURN-VALUE function. If the error is raised by executing a RETURN ERROR with an optional ABL error object or by executing an UNDO, THROW, the System.Exception Message property includes both a description of the operation where the error occurred and any messages from the ABL error object. If the error is fatal, the AVM responds as for any ABL class, generating a protrace file and exiting the session.

See also

Assignment (=) statement, Class-based property access, Data types, DEFINE VARIABLE statement, DISPLAY statement, METHOD statement, RETURN statement, VAR statement

DEFINE QUERY statement

Defines a query that is created at compile time for use in one or more procedures, or within a single class or class hierarchy. A query can be opened with an OPEN QUERY statement, and records can be retrieved using a GET statement, BROWSE widget, or the FILL( ) method on a ProDataSet object handle.

Syntax

```
DEFINE {[[ NEW ] SHARED ]|[ PRIVATE | PROTECTED ][ STATIC ]}
  QUERY query
  FOR buffer-name[field-list][ , buffer-name[field-list]]...
  [ CACHE n]
  [ SCROLLING ]
  [ RCODE-INFORMATION ]
```

`NEW SHARED QUERY` *query*

Defines and identifies a query to be shared with one or more procedures called directly or indirectly by the current procedure. The called procedures must define the same query name as SHARED. For shared queries, each *buffer-name* must be the name of a shared buffer. The shared buffers must be specified in the same order both across shared queries and in the OPEN QUERY.

`SHARED QUERY` *query*

Defines and identifies a query that was initially defined by another procedure as NEW SHARED. For shared queries, each *buffer-name* must be the name of a shared buffer. The shared buffers must be specified in the same order across shared queries and in the OPEN QUERY.

`[ PRIVATE | PROTECTED ][ STATIC ] QUERY` *query*

Defines and identifies a query as a data member of a class, and optionally specifies an access mode (PRIVATE or PROTECTED) and scope (instance or STATIC) for that data member. You cannot specify these options when defining a query as a data element of a method (including constructors, destructors, and property accessors) or a procedure. For a data member query, each *buffer-name* must be the name of a compatible data member buffer (see the FOR option).

Note: The specified options are applicable only when defining a data member for a class in a class definition (`.cls`) file. Note also that you cannot shadow (override) the definition of a given query data member in a class hierarchy.

PRIVATE query data members can be accessed by the defining class. An instance can access a private data member of another instance if both instances are from the same class. PROTECTED query data members can be accessed by the defining class and any of its derived classes. An instance can access a protected data member of a second instance that is at the same level or higher in the class hierarchy. The default access mode is PRIVATE. When you reference a query from another data member definition (such as a data-source) defined in the same class or class hierarchy, the access mode of the query cannot be more restrictive than the access mode of the referencing data member.

A query defined with the STATIC option is a static data member of the class type for which it is defined and is scoped to the ABL session where it is referenced. ABL creates one copy of the specified class static query on first reference to the class type, and ABL creates only one such copy for any number of instances of the class that you create. You can directly reference an accessible static query data member from any other static or instance class member defined in the same class or class hierarchy.

Without the STATIC option, ABL creates an instance query data member that is scoped to a single instance of the class where it is defined. ABL creates one copy of the specified instance query for each such class instance that you create. You cannot directly reference an instance query data member from a STATIC class member definition defined within the same class or class hierarchy.

For more information on accessing queries of different access modes and scopes, see the reference entry for Class-based data member access.

Note: Members of a class are grouped into six namespaces, including buffers/temp-tables, methods, variables/properties/events, ProDataSets, queries, and data-sources. Queries defined as members of a class share the same namespace. There can be only one class member in this namespace with a given name.

For more information on where and how to define data members in a class, see the CLASS statement reference entry.

`QUERY` *`query`*

Defines and identifies a query whose records you can access only within the current procedure, method of a class (including constructors, destructors, and property accessors), or as a PRIVATE data member of a class.

`FOR` *`buffer-name`* **`[`** *`field-list`* **`]`** **`[`** `,` *`buffer-name`* **`[`** *`field-list`* **`]`** **`]`** **`...`**

Specifies the buffers to be used by the query, where *`buffer-name`* is a table or alternate buffer name. For a shared query, each *`buffer-name`* must be a shared buffer. If the query is a data member of a class, you must specify the name of a compatible buffer. Thus, if the query is a static data member, the buffer must also be a static data member; if the query is PROTECTED, the buffer must also be defined as PROTECTED or inherited from a super class; and if the data-source is PRIVATE, the buffer can be defined as either PRIVATE or PROTECTED. If the query is a PRIVATE instance data member, you can also specify a default database table buffer; you cannot specify a default database buffer for a query data member defined with any other combination of access mode and scope.

Note: ABL defines the default buffer for every database table that you access in a class as a PRIVATE instance buffer data member.

If the query definition references more than one buffer, it defines a join.

Once the query has been defined, you cannot change the buffers that it references, even if the query is closed and re-opened. For example, a buffer, buff1, is created for the Customer table in a DEFINE QUERY or OPEN QUERY for the query, qry1. The query is run and closed. You cannot now DEFINE or OPEN qry1 with buff1 for the item table. You can reuse buffers with CREATE QUERY, but you must re-run QUERY-PREPARE.

The *field-list* is an optional list of fields to include or exclude when you open the query. Field lists allow you to retrieve only specified fields rather than all the fields of a record. They can improve performance over a network by enabling more records to fit into a single network message. This is the syntax for *field-list*:

```
{
    FIELDS [ ( [field...] ) ]
  | EXCEPT [ ( [field...] ) ]
}
```

The FIELDS option specifies the fields you want to include in the query, and the EXCEPT option specifies the fields that you want to exclude from the query. The *field* parameter is the name of a single field in the table specified by *buffer-name*. If *field* is an array reference, the whole array is retrieved. Specifying FIELDS with no *field* references retrieves no fields. However the ROWID value for a specified record is available using the ROWID function. Specifying EXCEPT with no *field* references or specifying *record* without a *field-list* causes the AVM to retrieve a complete record.

Note that the AVM ignores the FIELDS option for temp-tables.

Note:

Field lists should be used with caution because they can cause unexpected run-time errors. For example, it is possible that you did not include a field that is required by a new or revised subroutine or trigger. The result can be a run-time error that may be difficult to debug. A workaround is to use either the `-fldisable` or the `-rereadfields` client session startup parameter. This will allow your appplication to run (although more slowly) until the application can be fixed.

Note the difference between `-fldisable` and `-rereadfields`. The Field List Disable (`-fldisable`) parameter causes the AVM to ignore **all** field lists and to fetch the entire record for **every** query. The Reread Fields (`-rereadfields`) parameter causes the AVM to fetch the record with the specified fields, but then re-fetch the entire record if an ABL statement tries to reference a field that is not there. Also note that when the re-fetch happens, any `FIND` trigger is not run again. For more information on these startup parameters, see *Startup Command and Parameter Reference*.

This statement defines a query to retrieve only the name and balance fields from the Customer table:

```
DEFINE QUERY custq FOR Customer FIELDS (name balance).
```

When you specify a field list, the AVM might retrieve additional fields or the complete record depending on the type of retrieval operation and the DataServer that provides the record. Thus, the AVM:

- Retrieves any additional fields required by the client to complete the record selection.
- Retrieves a complete record when the record is fetched with EXCLUSIVE-LOCK. This ensures proper operation of updates and the local before-image (BI) file. For information on the local BI file, see *Manage the OpenEdge Database*.
- Retrieves a complete record for DataServers that do not support SHARE-LOCK. For more information, see the appropriate DataServer guides (*Use the Microsoft SQL Data Server* and *Use the Oracle Data Server*).

Note: Always specify fields that you plan to reference in the field list. Only those extra fields that the client requires for record selection are added to the specified field list. The AVM distributes record selection between the client and server depending on a number of factors that change with each OpenEdge release. Therefore, never rely on fields that you did not specify but which the AVM fetches for its own needs; they might not always be available. There is no additional cost to specify a field in the list that you otherwise expect the AVM to provide.

This statement defines a query to retrieve all fields of the Customer table except the name and balance fields:

```
DEFINE QUERY custq FOR Customer EXCEPT (name balance).
```

`CACHE` *n*

Specifies the number of records of the query to hold in memory for a NO-LOCK query. Generally, caching more records produces better browse performance when accessing a database across a network. However, caching consumes both memory and CPU time for buffer management.

If you specify the CACHE option, the SCROLLING option is assumed. If a query is referenced in a DEFINE BROWSE statement, caching occurs by default. The default for a query involving only one table is 50 records. The default for a multi-table query is 30 records. If you specify CACHE 0 in the DEFINE QUERY statement, no caching occurs.

`SCROLLING`

Specifies that you can jump to a location within the list of records that satisfy the query by using the REPOSITION statement. If you do not use this option, you can use only the FIRST, NEXT, LAST, and PREV options of the GET statement to navigate within the list.

The SCROLLING option can improve NETWORK performance because query NO-LOCK records can be prefetched with multiple records per network message. (Note that in a join only the lowest level records for a given set of upper level records, can be prefetched). The `-defaultscrolling` startup parameter has the same effect. See *Startup Command and Parameter Reference* for more information.

Non-network queries are faster if you do not use this option, but you must specify it to use the REPOSITION statement. If you do not specify SCROLLING for non-OpenEdge databases, you can only move forward through the list of records using the FIRST and NEXT options of the GET statement.

`RCODE-INFORMATION`

Note: This option is the default behavior (thus, it has no effect). It is supported only for backward compatibility.

Examples

The following example defines two queries, q-salesrep and q-cust. The first is opened in the main procedure block and is used to find all `SalesRep` records. The q-cust query is used to find all `Customers` associated with a `SalesRep`. The results of the q-cust query are displayed in a browse widget. The q-cust query is reopened each time you find a new `SalesRep`.

r-defqry.p

```
DEFINE QUERY q-salesrep FOR SalesRep
DEFINE QUERY q-cust FOR Customer

DEFINE BROWSE cust-brws QUERY q-cust
  DISPLAY Customer.CustNum Customer.Name Customer.Phone
    WITH 5 DOWN TITLE "Customer Information".

DEFINE BUTTON b_next LABEL "Next".
DEFINE BUTTON b_quit LABEL "Quit" AUTO-ENDKEY.

FORM
  SalesRep.SalesRep SalesRep.RepName SalesRep.Region SaleRep.MonthQuota
  WITH FRAME rep-info SIDE-LABELS TITLE "Sales Rep. Info".

FORM b_next space(5) b_quit
  WITH FRAME butt-frame COLUMN 60.

ON CHOOSE OF b_next DO:
  GET NEXT q-salesrep.
  IF NOT AVAILABLE SalesRep THEN GET FIRST q-salesrep.
  RUN disp-rep.
END.

OPEN QUERY q-salesrep FOR EACH SalesRep NO-LOCK.

GET FIRST q-salesrep.
RUN disp-rep.

ENABLE cust-brws WITH FRAME cust-info.
ENABLE ALL WITH FRAME butt-frame.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.

PROCEDURE disp-rep:
  DISPLAY SalesRep.SalesRep Salesrep.RepName SalesRep.Region
    SalesRep.MonthQuota
    WITH FRAME rep-info CENTERED SIDE-LABELS TITLE "Sales Rep. Info".
  OPEN QUERY q-cust FOR EACH Customer OF SalesRep NO-LOCK.
END PROCEDURE.
```

The following example uses the RCODE-INFORMATION option of the DEFINE QUERY statement to extract index information from a compile-time defined query. If you run the example with the RCODE-INFORMATION option commented out, the AVM reports a run-time error.

r-rcdinf.p

```
/* r-rcdinf.p */
/* Extracts index information from a compile-time defined query.*/
DEFINE VARIABLE h AS HANDLE NO-UNDO.

DEFINE QUERY q FOR Customer RCODE-INFORMATION.

h = QUERY q:HANDLE.
OPEN QUERY q FOR EACH Customer BY Customer.Name.

MESSAGE h:INDEX-INFORMATION.
```

Notes

- You cannot define a SHARED or NEW SHARED query in a class definition (`.cls`) file. If you do, ABL generates a compilation error.

- A query can be compile-time defined (often referred to as a *static* query object), where the query is defined and created at compile time using this statement, or it can be run-time defined (often referred to as a *dynamic* query object), where the query is defined and created at run time using the CREATE QUERY statement and query object handle operations. A compile-time defined query can also be defined as a static data member of a class. In this case, it is a static query object that is also a class static data member.
- After you define a query, you must open it with the OPEN QUERY statement before you can fetch any records.
- A SHARED query remains in scope for an instance of a persistent procedure until the instance is deleted. This is true even if the original procedure that defined the query as NEW SHARED goes out of scope while the procedure instance remains persistent.

  If a trigger or internal procedure of a persistent procedure executes an external subprocedure that defines a SHARED query, ABL includes the persistent procedure in the resolution of the corresponding NEW SHARED query as though the procedure were on the procedure call stack.
- Unlike with block record retrieval operations that include record updates and deletes (FOR EACH, etc.), field lists generally enhance query performance even for queries whose rows you plan to update. Queries generate complete result lists, with or without field lists, before any updates to individual rows are applied.
- You cannot specify field lists in an OPEN QUERY statement.
- In a shared query, the shared buffers must be specified in the same order across all the shared queries and in the OPEN QUERY statement.
- A ProDataSet data-relation defined with REPOSITION or SELECTION is overridden if a query is defined for a child relation data-source. Normally, if REPOSITION is not specified, the child query selects the children of the parent. But coding a query for the child data-source overrides whether or not a SELECTION or REPOSITION relation mode was defined.

See also

Class-based data member access, CLOSE QUERY statement, CREATE QUERY statement, CURRENT-RESULT-ROW function, DEFINE BROWSE statement, DEFINE DATASET statement, GET statement, NUM-RESULTS function, OPEN QUERY statement, REPOSITION statement, RUN statement

DEFINE RECTANGLE statement

Defines a rectangle widget that is created at compile time for use within a single procedure or class.

Note: Does not apply to SpeedScript programming.

Syntax

```
DEFINE [ PRIVATE ] RECTANGLE rectangle[ LIKE rectangle2]
  [ NO-FILL ]
  [{ EDGE-CHARS width}|{ EDGE-PIXELS width}]
  [ DCOLOR expression]
  [ BGCOLOR expression]
  [ FGCOLOR expression]
  [ GRAPHIC-EDGE ]
  [ PFCOLOR expression]
  [ ROUNDED ]
  [ GROUP-BOX ]
  [size-phrase]
  [ TOOLTIP tooltip]
  {[trigger-phrase]}
```

[PRIVATE] RECTANGLE *rectangle*

Defines and identifies a rectangle widget as a class-scoped object. A class-scoped handle-based object is not a member of a class, but provides a resource that is privately available to the class definition similar to a non-shared data element in a procedure definition. The option to specify the PRIVATE access mode is provided for readability. You cannot specify PRIVATE when defining a rectangle widget as a data element in a method or procedure.

Note: This option is applicable only when defining a class-scoped rectangle widget in a class definition (`.cls`) file.

`RECTANGLE` *rectangle*

Defines and identifies a rectangle widget for access only within the current procedure, class, or method of a class.

`LIKE` *rectangle2*

Specifies a previously defined rectangle whose characteristics you want to apply to the new rectangle. If you name a rectangle with this option, you must have defined that rectangle previously in the procedure.

`NO-FILL`

Indicates that only the outline of the rectangle should be drawn. By default, the rectangle is filled with the background color.

`EDGE-CHARS` *width*

Specifies the width of the rectangle outline in characters. The default width is 1. If you do not want an edge on the rectangle, specify EDGE-CHARS 0.

EDGE-PIXELS *width*

Specifies the width of the rectangle outline in pixels. The default width is 1. If you do not want an edge on the rectangle, specify EDGE-PIXELS 0.

DCOLOR *expression*

Specifies the fill color of the rectangle in character interfaces. This option is ignored in graphical interfaces.

BGCOLOR *expression*

Specifies the background color or fill color of the rectangle in graphical interfaces. This option is ignored in character interfaces.

FGCOLOR *expression*

Specifies the foreground color or edge color of the rectangle in graphical interfaces. This option is ignored in character interfaces.

GRAPHIC-EDGE

Specifies that in a character interface, the rectangle is drawn with graphic characters. This option is ignored in a graphical interface. This overrides the EDGE-CHARS and EDGE-PIXELS options. The border is one graphic unit thick.

PFCOLOR *expression*

Specifies the edge color of the rectangle in character interfaces. This option is ignored in graphical interfaces. It is also ignored if you specify GRAPHIC-EDGE.

ROUNDED

Specifies that the rectangle appears with rounded corners (whether the display is set to the Windows XP Theme or the Windows Classic Theme).

This option is ignored in character interfaces.

GROUP-BOX

Specifies the rectangle as a group box, which surrounds one or more user interface widgets to visually indicate a relationship among the widgets. The appearance of a group box changes automatically to conform to the current display setting. When the display is set to the Windows XP Theme, the group box rectangle appears with rounded corners and a single-line border. When the display is set to the Windows Classic Theme, the group box appears with square corners and a shaded double-line border.

This option is ignored in character interfaces.

size-phrase

Specifies the outside dimensions of the rectangle widget. This is the syntax for *size-phrase*:

Syntax

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS }width BY height
```

If you specify SIZE or SIZE-CHARS, the units are characters; if you specify SIZE-PIXELS, the units are pixels. For character units, the values *width* and *height* must be decimal constants. For pixels units, they must be integer constants. For more information, see the SIZE phrase reference entry.

`TOOLTIP` *tooltip*

Allows you to define a help text message for a rectangle widget. The AVM automatically displays this text when the user pauses the mouse button over the rectangle widget.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the ToolTip is removed. No ToolTip is the default. The TOOLTIP option is supported in Windows only.

trigger-phrase

Specifies application triggers for the rectangle.

For more information, see the Trigger phrase reference entry.

Example

The following example uses a set of thin rectangles as lines to create graphic columns within a frame background:

r-bkgrnd.p

```
DEFINE VARIABLE item-tot AS DECIMAL NO-UNDO LABEL "Value".

DEFINE RECTANGLE vline1 SIZE .4 BY 5 EDGE-PIXELS 2.
DEFINE RECTANGLE vline2 LIKE vline1.
DEFINE RECTANGLE vline3 LIKE vline1.
DEFINE RECTANGLE vline4 LIKE vline1.
DEFINE RECTANGLE vline5 LIKE vline1.
DEFINE RECTANGLE vline6 LIKE vline1.
DEFINE RECTANGLE hline SIZE 78 BY .1 EDGE-PIXELS 2.

DEFINE FRAME item-info
  Item.ItemNum
  Item.ItemName
  Item.OnHand
  Item.ReOrder
  Item.OnOrder
  Item.Price
  item-tot
  BACKGROUND SKIP(1) hline
    vline1 AT 9
    vline2 AT 25
    vline3 AT 33
    vline4 AT 42
    vline5 AT 51
    vline6 AT 65
  WITH TITLE "Inventory Current Value" CENTERED USE-TEXT 5 DOWN.

FOR EACH Item NO-LOCK WITH FRAME item-info:
  DISPLAY
    Item.ItemNum
    Item.ItemName
    Item.OnHand
    Item.ReOrder
    Item.OnOrder
    Item.Price
    Item.OnHand * Item.Price @ item-tot.
```

Notes

- To create the compile-time defined rectangle you are defining, you must define a compile-time defined frame that contains the rectangle. Each frame you define that contains the same rectangle creates an additional instance of that rectangle. The handle for a compile-time defined rectangle is not available until the rectangle is created.
- You can specify an application-defined widget ID for a compile-time defined rectangle widget using the *form-item* phrase in either the FORM statement or the DEFINE FRAME statement. See the FORM statement and DEFINE FRAME statement reference entries for more information.
- When defining a rectangle, you must specify either the LIKE option or the size phrase.

See also

Class-based data member access, FORM statement

DEFINE STREAM statement

Defines a stream for use in one or more procedures, or within a single class. Use this statement when you want to use streams other than the two ABL built-in unnamed streams. Using additional streams allows you to get input from more than one source simultaneously or to send output to more than one destination simultaneously.

Syntax

```
DEFINE { [ [ NEW [ GLOBAL ] ] SHARED ] | [ PRIVATE ] } STREAM stream-name
```

`NEW SHARED STREAM` *stream-name*

Defines and identifies a stream that can be shared by other procedures. When the procedure using the DEFINE NEW SHARED STREAM statement ends, the stream is no longer available to any procedure.

`NEW GLOBAL SHARED STREAM` *stream-name*

Defines and identifies a stream that can be shared by other procedures and that will remain available even after the procedure that contains the DEFINE NEW GLOBAL SHARED STREAM statement ends.

`SHARED STREAM` *stream-name*

Defines and identifies a stream that was created by another procedure using the DEFINE NEW SHARED STREAM statement or the DEFINE NEW GLOBAL SHARED STREAM statement.

`[ PRIVATE ] STREAM` *stream-name*

Defines and identifies a stream as a class-scoped object. A class-scoped handle-based object is not a member of a class, but provides a resource that is privately available to the class definition similar to a non-shared data element in a procedure definition. The option to specify the PRIVATE access mode is provided for readability. You cannot specify PRIVATE when defining a stream as a data element in a method or procedure.

Note: This option is applicable only when defining a class-scoped stream in a class definition (`.cls`) file.

STREAM *stream-name*

Defines and identifies a stream for access only within the current procedure, class, or method of a class.

Example

This procedure, in a single pass through the item table, uses the rpt stream to create a report and the exceptions stream to create a list of exceptions:

r-dfstr.p

```
DEFINE VARIABLE fnr     AS CHARACTER NO-UNDO FORMAT "x(12)".
DEFINE VARIABLE fne     AS CHARACTER NO-UNDO FORMAT "x(12)".
DEFINE VARIABLE excount AS INTEGER   NO-UNDO
  LABEL "Total Number of exceptions".
DEFINE NEW SHARED BUFFER xitem FOR item.

DEFINE NEW SHARED STREAM rpt.
DEFINE STREAM exceptions.

SET fnr LABEL "Enter filename for report output" SKIP(1)
  fne LABEL "Enter filename for exception output"
  WITH SIDE-LABELS FRAME fnames.

OUTPUT STREAM rpt TO VALUE(fnr) PAGED.
OUTPUT STREAM exceptions TO VALUE(fne) PAGED.

FOR EACH xitem:
  IF on-hand < alloc THEN DO:
    DISPLAY STREAM exceptions
      xitem.ItemNum xitem.ItemName xitem.OnHand xitem.Allocated
      WITH FRAME exitem DOWN.
    excount = excount + 1.
  END.
  RUN r-dfstr2.p.
END.

DISPLAY STREAM exceptions SKIP(1) excount WITH FRAME exc SIDE-LABELS.
DISPLAY STREAM rpt WITH FRAME exc.

OUTPUT STREAM rpt CLOSE.
OUTPUT STREAM exceptions CLOSE.
```

Include the DISPLAY statement in the `r-dfstr2.p` procedure in the `r-dfstr.p` procedure for efficiency. (It is in a separate procedure here to illustrate shared streams.)

r-dfstr2.p

```
DEFINE SHARED STREAM rpt.
DEFINE SHARED BUFFER xitem FOR Item.

DISPLAY STREAM rpt ItemNum ItemName WITH NO-LABELS NO-BOX.
```

Notes

- You cannot define a SHARED or NEW SHARED stream in a user-defined function, an internal procedure, or a persistent procedure. If you do, the AVM raises an ERROR on the RUN statement that creates the procedure.
- You cannot define a SHARED or NEW SHARED stream in a class definition (`.cls`) file. If you do, ABL generates a compilation error.
- You can overcome the limitations on SHARED or NEW SHARED streams by using stream handles. For more information, see the Stream object handle reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces*.
- ABL automatically provides two unnamed streams to each procedure: the input stream and the output stream. These streams give the procedure a way to communicate with an input source and an output destination. For example, the following statement tells ABL to use the unnamed input stream to get input from the file named `testfile`:

```
INPUT FROM testfile.
```

- Using the DEFINE STREAM statement creates a stream, but it does not actually open that stream. To open a stream, you must use the STREAM option with the INPUT FROM, INPUT THROUGH, OUTPUT TO, OUTPUT THROUGH, or INPUT-OUTPUT THROUGH statements. You must also use the STREAM option with any data handling statements that move data to and from the stream.
- After you open the stream, you can use the SEEK function to return the offset value of the file pointer, or you can use the SEEK statement to position the file pointer to any location in the file.
- For information about limits on the number of streams per procedure, see the application limits appendix in *Manage ABL Applications*.

See also

Class-based data member access, DISPLAY statement, INPUT CLOSE statement, INPUT FROM statement, INPUT THROUGH statement, INPUT-OUTPUT THROUGH statement, OUTPUT CLOSE statement, OUTPUT THROUGH statement, OUTPUT TO statement, PROMPT-FOR statement, RUN statement, SEEK function, SEEK statement, SET statement, Stream object handle

DEFINE SUB-MENU statement

Defines a submenu widget that is created at compile time for use within a single procedure or class. You can use a submenu widget as a pull-down menu within a menu bar or as a submenu of a pull-down menu or pop-up menu.

Note: Does not apply to SpeedScript programming.

Syntax

```
DEFINE [ PRIVATE ] SUB-MENU submenu
   [ BGCOLOR expression ]
   [ DCOLOR expression ]
   [ FGCOLOR expression ]
   [ PFCOLOR expression ]
   [ FONT number ]
   [ SUB-MENU-HELP ]
   { LIKE menu | menu-element-descriptor . . . }
```

[PRIVATE **]** SUB-MENU *submenu*

Defines and identifies a sub-menu widget as a class-scoped object. A class-scoped handle-based object is not a member of a class, but provides a resource that is privately available to the class definition similar to a non-shared data element in a procedure definition. The option to specify the PRIVATE access mode is provided for readability. You cannot specify PRIVATE when defining a sub-menu widget as a data element in a method or procedure.

Note: This option is applicable only when defining a class-scoped sub-menu widget in a class definition (`.cls`) file.

SUB-MENU *submenu*

Defines and identifies a submenu for access only within the current procedure, class, or method of a class.

BGCOLOR *expression*

Specifies the background color for the submenu in graphical interfaces. This option is ignored in character interfaces and Windows.

DCOLOR *expression*

Specifies the display color for the submenu in character interfaces. This option is ignored in graphical interfaces.

FGCOLOR *expression*

Specifies the foreground color for the submenu in graphical interfaces. This option is ignored in character interfaces.

PFCOLOR *expression*

Specifies the prompt-for color for the submenu in character interfaces. This option is ignored in graphical interfaces.

FONT *number*

Has no effect; supported only for backward compatibility.

SUB-MENU-HELP

Has no effect; supported only for backward compatibility.

LIKE *menu*

Specifies a previously defined menu or submenu whose characteristics you want to apply to the new submenu. If you name a menu with this option, you must have previously defined that menu in the procedure. If you name a submenu with this option, that submenu must have already been used as part of a menu definition.

menu-element-descriptor

Specifies an element displayed on the menu. Each element is either a choosable menu item, a submenu, non-choosable text, a rule, or a blank space. You must specify one or more menu elements, unless you use the LIKE option.

This is the syntax for *menu-element-descriptor*:

```
{
  RULE
  SKIP
  SUB-MENU submenu [ DISABLED ] [ LABEL label ]
  menu-item-phrase
}
```

RULE

Specifies that a rule or line is inserted at this point in the submenu. You can use this, for example, to divide the submenu into sections.

SKIP

Specifies that a blank line is inserted at this point in the submenu. You can use this, for example, to divide the submenu into sections.

SUB-MENU *submenu* [DISABLED] [LABEL *label*]

Specifies that a submenu is displayed at this menu item. The submenu must be previously defined in the procedure. The submenu appears when the user chooses that item. The submenu cannot be a menu bar. The DISABLED and LABEL options for a submenu are the same as described for the *menu-item-phrase*.

menu-item-phrase

Specifies a choosable menu item. This is the syntax for *menu-item-phrase*:

```
MENU-ITEM menu-item-name
   [ ACCELERATOR keylabel ]
   [ BGCOLOR expression ]
   [ DCOLOR expression ]
   [ DISABLED ]
   [ FGCOLOR expression ]
   [ FONT expression ]
   [ LABEL label ]
   [ PFCOLOR expression ]
   [ READ-ONLY ]
   [ TOGGLE-BOX ]
  { [ trigger-phrase ] }
```

MENU-ITEM *menu-item-name*

The name of the menu item you are defining.

ACCELERATOR *keylabel*

Specifies a keyboard accelerator for this menu item. A keyboard accelerator is a key—possibly modified by **SHIFT**, **CONTROL**, or **ALT**—that chooses a menu item even if the menu is not displayed. The value *keylabel* must be a character-string expression that evaluates to a valid key label recognized by the AVM, such as a, **F1**, or **ALT+SHIFT+F1**. See the topics on handling user input in the *OpenEdge Programming Interfaces* for the precedence rules that the AVM uses for handling keyboard input.

BGCOLOR *expression*

Specifies the background color for the menu item in graphical interfaces. If you omit this option, the menu item inherits the background color of the submenu.

DCOLOR *expression*

Specifies the display color for the menu item in character interfaces. If you omit this option, the menu item inherits the display color of the submenu.

DISABLED

Specifies that the menu item is initially disabled for input. This means that the user cannot choose this item. Disabled items are grayed out in environments that support it.

FGCOLOR *expression*

Specifies the foreground color for the menu item in graphical interfaces. If you omit this option, the menu item inherits the foreground color of the submenu.

FONT *expression*

Specifies the font for the menu item. If you omit this option, the menu item inherits the font of the menu.

LABEL *label*

Specifies the text that displays in the submenu for a choosable menu item or submenu. If you omit LABEL, the AVM displays the item handle by default.

You can include an ampersand (&) within the label to indicate that the following letter acts as a mnemonic for the menu item. This means that when the menu is displayed, the user can choose the item by pressing that single key. If you do not include an ampersand within the label, Windows treats the first character as a mnemonic. To include a literal ampersand within a label, specify a double ampersand (&&).

PFCOLOR *expression*

Specifies the prompt-for color for the menu item in character interfaces. If you omit this option, the menu item inherits the prompt-for color of the submenu.

READ-ONLY

Specifies that this menu item is read-only text. The user cannot choose this item.

TOGGLE-BOX

Specifies that the menu item is displayed as a checkbox that the user can toggle on or off. In environments that do not support this option, it is ignored.

trigger-phrase

Specifies application triggers for the menu item. Typically, you associate a CHOOSE trigger with each menu item.

For more information, see the Trigger phrase reference entry.

Example

The r-menu.p procedure defines three pull-down submenus. One of the submenus, myedit, contains a nested submenu, myobjects. The procedure defines a menu bar, mybar, that contains two submenus labelled File and Edit. The handle of mybar is assigned to a window mywin. The ON statements define triggers to execute when you choose the corresponding menu items.

r-menu.p

```
DEFINE VARIABLE mywin AS HANDLE NO-UNDO.

DEFINE SUB-MENU myfile
  MENU-ITEM m1  LABEL "Save"
  MENU-ITEM m2  LABEL "Save As"
  MENU-ITEM m3  LABEL "Exit".

DEFINE SUB-MENU myobjects
  MENU-ITEM m1  LABEL "Circle"
  MENU-ITEM m2  LABEL "Line"
  MENU-ITEM m3  LABEL "Rectangle"
  MENU-ITEM m4  LABEL "Text".

DEFINE SUB-MENU myedit
  SUB-MENU myobjects LABEL "Add"
  MENU-ITEM e1       LABEL "Delete"
  MENU-ITEM e2       LABEL "Copy".

DEFINE MENU mybar MENUBAR
  SUB-MENU myfile LABEL "File"
  SUB-MENU myedit LABEL "Edit".

CREATE WINDOW mywin
  ASSIGN MENUBAR = MENU mybar:HANDLE.

DEFINE BUTTON b1 LABEL "Text Mode".
DEFINE BUTTON b2 LABEL "Graphics Mode".

CURRENT-WINDOW = mywin.
FORM
  b1 at X 10  Y 120
  b2 at x 120 Y 120
  WITH FRAME x.
ENABLE b1 b2 WITH FRAME x.

ON CHOOSE OF b1 IN FRAME x DO:
  MENU-ITEM m1:SENSITIVE IN MENU myobjects = FALSE.
  MENU-ITEM m2:SENSITIVE IN MENU myobjects = FALSE.
  MENU-ITEM m3:SENSITIVE IN MENU myobjects = FALSE.
  MENU-ITEM m4:SENSITIVE IN MENU myobjects = TRUE.
END.

ON CHOOSE OF b2 IN FRAME x DO:
  MENU-ITEM m1:SENSITIVE IN MENU myobjects = TRUE.
  MENU-ITEM m2:SENSITIVE IN MENU myobjects = TRUE.
  MENU-ITEM m3:SENSITIVE IN MENU myobjects = TRUE.
  MENU-ITEM m4:SENSITIVE IN MENU myobjects = FALSE.
END.

WAIT-FOR CHOOSE OF MENU-ITEM m3 IN MENU myfile.
DELETE WIDGET mywin.
```

Notes

- To create the compile-time defined submenu you are defining, along with any of its descendents (submenus and menu items), you must define a compile-time defined menu that contains the submenu. Each menu you define that contains the same submenu creates an additional instance of the submenu and each of its descendents. The handles for a compile-time defined submenu and its descendents are not available until the submenu is created in a menu.
- You cannot define a submenu with the same name more than once in the same menu tree. Thus, if menu mFile contains both submenu mOptions and submenu mSave, submenu mSave cannot also contain submenu mOptions.

- Menu items in different menus and submenus can have the same names. In the above procedure, the menu items in myfile and myobjects share the same names. To avoid ambiguity, use the IN MENU or IN SUB-MENU option to identify the parent menu or submenu.
- There are instances where you cannot avoid ambiguity in menu item references. In such instances, ABL always references the first unambiguous instance of the menu item. In particular, if the same submenu containing a menu item appears in more than one menu and each menu defines another instance of the same menu item, you can only reference that menu item in the submenu from the **first** menu that contains it. Thus, if submenu mOptions contains menu item mSave and the menus mFile and mDraw (in that order) both contain submenu mOptions and another menu item mSave, you can only reference menu item mSave in submenu mOptions from menu mFile. You cannot uniquely reference menu item mSave in submenu mOptions from menu mDraw because menu mDraw contains another menu item mSave.
- When a menu item is disabled, it appears grayed-out (if the environment supports that) and it cannot be chosen.

See also

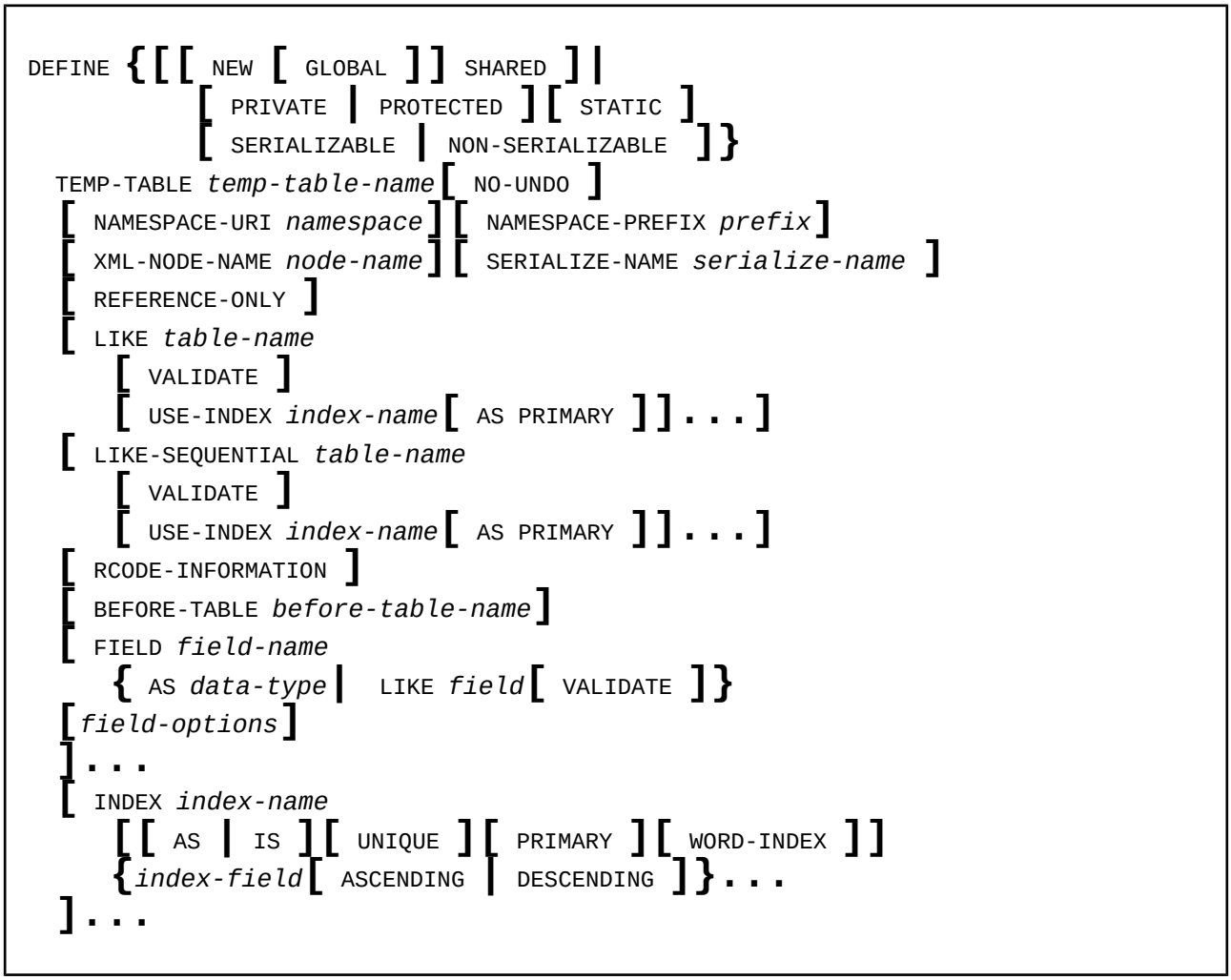
Class-based data member access, CREATE widget statement, Trigger phrase

---

DEFINE TEMP-TABLE statement

Defines a temp-table that is created at compile time. The AVM stores temp-tables in memory (with potential overflow to disk). Among procedures, a temp-table can be either global (lasting for the entire ABL session) or local (lasting only as long as the procedure that creates it), and either shared (visible to other procedures that want to access it) or non-shared (visible just to the procedure that created it). In a class, a temp-table can be defined for use within a single class or class hierarchy.

Syntax



`NEW SHARED TEMP-TABLE` *`temp-table-name`*

Defines and identifies a temp-table object that can be shared by one or more procedures called directly or indirectly by the current procedure. The temp-table remains available to other procedures until the procedure that defined it ends. The called procedures must define the same temp-table name using a DEFINE SHARED TEMP-TABLE statement.

Note: A SHARED temp-table cannot have a BEFORE-TABLE.

`NEW GLOBAL SHARED TEMP-TABLE` *`temp-table-name`*

Defines and identifies a global shared temp-table object. The scope of a global shared temp-table is the ABL session. The first procedure to define a temp-table NEW GLOBAL SHARED establishes it. Subsequent procedures access it using a DEFINE SHARED TEMP-TABLE statement.

Note: ABL does not establish multiple global shared temp-tables with the same name in the same ABL session.

Note: A SHARED temp-table cannot have a BEFORE-TABLE.

SHARED TEMP-TABLE *temp-table-name*

Defines and identifies a temp-table object that was initially defined by another procedure using a DEFINE NEW SHARED TEMP-TABLE or DEFINE NEW GLOBAL SHARED TEMP-TABLE statement.

The procedure that establishes the temp-table determines the name. The procedures that share the temp-table use that name to identify it.

[PRIVATE | PROTECTED] [STATIC] [SERIALIZABLE | NON-SERIALIZABLE]
TEMP-TABLE *temp-table-name*

Defines and identifies a temp-table object as a data member of a class, and optionally specifies an access mode (PRIVATE or PROTECTED) and scope (instance or STATIC) for that data member, as well as whether or not the temp-table will participate in serialization (SERIALIZABLE or NON-SERIALIZABLE). You cannot specify any of these options for a temp-table in an interface definition (INTERFACE statement block) or when defining a temp-table as a data element of a procedure.

Note: The specified options are applicable only when defining a data member for a class in a class definition (`.cls`) file. The PRIVATE/PROTECTED modifier and the STATIC modifier can appear in either order, e.g., `DEFINE STATIC PRIVATE TEMP-TABLE myTempTable...` is valid syntax. Note also that you cannot shadow (override) the definition of a given temp-table data member in a class hierarchy.

The default access mode is PRIVATE.

A temp-table defined with the STATIC option is a static data member of the class type for which it is defined, and it is scoped to the ABL session where it is referenced. ABL creates one copy of the specified class static temp-table at the first reference to the class type, and creates only one such copy for any number of instances of the class that you create. You cannot specify STATIC if you specify the REFERENCE-ONLY option. You can directly reference an accessible static temp-table data member from any other static or instance class member defined in the same class or class hierarchy.

Without the STATIC option, ABL creates an instance temp-table data member that is scoped to a single instance of the class where it is defined. ABL creates one copy of the specified instance temp-table for each such class instance that you create. You cannot directly reference an instance temp-table data member from a STATIC class member definition defined within the same class or class hierarchy.

For more information on accessing temp-tables of different access modes and scopes, see the reference entry for Class-based data member access.

Class-based objects that are defined as serializable (using the SERIALIZABLE option in the CLASS statement) can be passed as parameters in remotes call between the application server and ABL clients and can be serialized to binary or JSON format. By default, both passing a class as a parameter and serializing an object to binary via the `Progress.IO.BinarySerializer` class include all data members regardless of access mode. However, for JSON serialization via `Progress.IO.JsonSerializer`, only public data members are serialized. To include a protected or private property during JSON serialization, SERIALIZABLE must be added to the definition. See Serialize( ) method (JsonSerializer) for more information.

Use the NON-SERIALIZABLE option to exclude a given temp-table from parameter passing between an application server and ABL client and from the serialization process via the `Progress.IO.BinarySerializer` class. (Temp-tables are NON-SERIALIZABLE by default via the `Progress.IO.JsonSerializer` class because they cannot be defined as public.) Fields of a temp-table marked as NON-SERIALIZABLE revert to their initial values when the class is deserialized.

Note: Members of a class are grouped into six namespaces, including buffers/temp-tables, methods, variables/properties/events, ProDataSets, queries, and data-sources. Buffers and temp-tables defined as members of a class share the same namespace. There can be only one class member in this namespace with a given name.

For more information on where and how to define data members in a class, see the CLASS statement reference entry.

`TEMP-TABLE` *`temp-table-name`*

Defines and identifies a temp-table object in an interface, or for access only within the current external procedure or as a data member of the current class.

`NO-UNDO`

Specifies that when a transaction is undone, changes to the temp-table records need not be undone. If you do not specify this option, all records in the temp-table are restored to their prior condition when a transaction is undone. The NO-UNDO option can significantly increase the performance for temp-table updates; use it whenever possible.

`NAMESPACE-URI` *`namespace`*

An optional CHARACTER constant that specifies the URI for the namespace of the temp-table.

`NAMESPACE-PREFIX` *`prefix`*

An optional CHARACTER constant that specifies the namespace prefix associated with the NAMESPACE-URI.

`XML-NODE-NAME` *`node-name`*

An optional CHARACTER constant that specifies the name of the XML element representing the temp-table in an XML Document. The default is *`temp-table-name`*. This option is useful when:

- reading a temp-table from XML with a name that contains invalid characters for an ABL name or is an ABL keyword, or
- writing a temp-table to XML with a name that is an invalid XML tag name.

Note: If you set SERIALIZE-NAME but do not set XML-NODE-NAME, the AVM sets XML-NODE-NAME equal to SERIALIZE-NAME.

`SERIALIZE-NAME` *`serialize-name`*

An optional CHARACTER constant that specifies the name of the element representing the temp-table in an XML Document or JSON string. The default is *`temp-table-name`*. This option is useful when:

- reading a temp-table from XML or JSON with a name that contains invalid characters for an ABL name or is an ABL keyword, or
- writing a temp-table to XML or JSON with a name that is an invalid XML tag name or an invalid JSON property name.

Note: If you set both XML-NODE-NAME and SERIALIZE-NAME, the READ-XML() and WRITE-XML() methods will use XML-NODE-NAME.

REFERENCE-ONLY

Specifies that the procedure defining this temp-table object is using the object definition only as a reference to a temp-table object that is defined and instantiated in another procedure or class, and specified as a parameter in the invocation of a RUN statement, a method in a class, or a user-defined function, using either the BY-REFERENCE or BIND option. The AVM does not instantiate the reference-only object. You cannot specify REFERENCE-ONLY if you specify the STATIC option.

Passing a reference-only temp-table object parameter to a local routine using either the BY-REFERENCE or BIND option allows the calling routine and the called routine to access the same object instance (instead of deep-copying the parameter).

Note: If you pass the parameter to a remote procedure, the AVM deep-copies the parameter on OUTPUT and the reference-only parameter is bound to that copy.

When you pass a temp-table parameter to a local routine using the BY-REFERENCE option, both the calling and called routines access the calling routine's object instance (and ignore the called routine's object instance). Since the called routine's object instance is ignored, you should define the object as a reference-only object. When you define a reference-only temp-table object in the called routine and receive it from the calling routine using the BY-REFERENCE option, the AVM binds the definition of the object in the called routine to the object instance in the calling routine for the duration of the called routine. You cannot define a reference-only temp-table object in the calling routine and pass it to the called routine using the BY-REFERENCE option.

When you pass a temp-table parameter to a local routine using the BIND option, you can define a reference-only temp-table object in either the calling routine or the called routine as follows:

- When you define a reference-only temp-table object in the calling routine and pass it to the called routine using the BIND option, the AVM binds the calling routine to the object instance in the called routine. The reference-only object definition remains bound to the object instance until the routine containing the reference-only object definition is deleted or terminates. The parameter must be an OUTPUT parameter.

  Note: If you also define the temp-table object instance in the called routine as a reference-only object, you must bind the object in the called routine before returning to the calling routine.

- When you define a reference-only temp-table object in the called routine and receive it from the calling routine using the BIND option, the AVM binds the called routine to the object instance in the calling routine. The reference-only object definition remains bound to the object instance until the routine containing the reference-only object definition is deleted or terminates. The parameter must be an INPUT or INPUT-OUTPUT parameter.

In either case, you must specify the BIND option for the parameter in both the invocation of a RUN statement, a method in a class, or a user-defined function, and in the DEFINE PARAMETER statement.

Caution: Do not delete the object or routine to which a reference-only temp-table object is bound, or you might be left with references to an object that no longer exists.

A reference-only temp-table object can be a member of a reference-only ProDataSet object or a standard ProDataSet object. However, if you define a reference-only temp-table in a standard ProDataSet object, you cannot use the ProDataSet object until you bind the reference-only temp-table.

LIKE *table-name* [USE-INDEX *index-name* [AS PRIMARY]] . . .

Specifies the name of a table whose characteristics the temp-table inherits. All field definitions of *table-name* are added to the temp-table. *table-name* can represent a database table or another temp-table.

Note: The source (temp-table or database table) for *table-name* can have any access mode or scope as long as its definition is accessible to the current temp-table definition.

If you reference a database field, the database containing that field must be connected at compile time. If the database field has a validation expression defined in the dictionary that contains a database reference, and the VALIDATE option is specified, the database must also be connected at run-time.

HELP options are inherited from the *table-name*. Validate options are inherited only if the VALIDATE keyword is used.

Some index definitions from the specified table might also be added to the temp-table:

- If you use the USE-INDEX option, only the definitions of indexes you specify with that option are copied to the temp-table. If one of these indexes is the primary index of the LIKE table, it becomes the default primary index of the temp-table. You can, however, use the AS PRIMARY option to override this default primary index.

  For example, to make the index country-post the primary index (thereby, overriding the default primary index `CustNum` in the table Customer), you specify it as follows:

```
DEFINE TEMP-TABLE mycust LIKE Customer
  USE-INDEX CustNum USE-INDEX country-post AS PRIMARY.
```

- If you do not specify the USE-INDEX option and do not use the INDEX option of the DEFINE TEMP-TABLE statement, then all index definitions are copied from the specified table to the temp-table. In this case, the primary index of the specified table becomes the primary index of the temp-table.
- If you do not specify the USE-INDEX option but do use the INDEX option of the DEFINE TEMP-TABLE statement, then no indexes are copied from the specified table unless the specified table is another temp-table.
- The AVM does not copy inactive indexes to the temp-table.
- If the source database table contains inactive indexes, then you must specify one or both of the USE-INDEX and INDEX options. If you do not, a compile time error is generated.

LIKE-SEQUENTIAL *table-name*

[USE-INDEX *index-name* [AS PRIMARY]] . . .

Specifies the name of a table whose characteristics the temp-table inherits. All field definitions of *table-name* are added to the temp-table. *table-name* can represent a database table or another temp-table.

LIKE-SEQUENTIAL is similar to LIKE in all ways except one. Unlike LIKE, which creates temp-table fields in _field._field-rpos order (POSITION order in the `.df` schema definition file) of the source table's fields, LIKE-SEQUENTIAL creates fields in _field._order sequence.

You can guarantee agreement of temp-table field order between any client and any application server using LIKE-SEQUENTIAL, as long as the _field._order values are the same. LIKE-SEQUENTIAL uses the field order as defined in the Data Dictionary when the source is a database table.

Note: The original behavior of LIKE was used to support RAW-TRANSFER with temp-tables. If you are using RAW-TRANSFER between a database table and a temp-table defined LIKE the database table, then you should not use LIKE-SEQUENTIAL.

`VALIDATE`

The temp-table fields inherit, from the dictionary, validation expressions and validation messages from the database table, *`table-name`*.

`RCODE-INFORMATION`

This option is supported only for backward compatibility.

`BEFORE-TABLE` *`before-table-name`*

Specifies the name of the before-image table associated with a compile-time defined temp-table in a ProDataSet object. You must specify a before-image table name for any compile-time defined ProDataSet temp-table for which you want to track changes. If you try to modify the records in this before-image table, the AVM generates a run-time error. You cannot use this option on a SHARED temp-table.

`FIELD` *`field-name`*

Defines a field in the temp-table. You can use FIELD clauses with the LIKE option to define additional fields for the temp-table, or you can define all your fields with FIELD clauses.

`AS` *`data-type`*

Specifies the data type of the field. The valid data types are BLOB, CHARACTER, CLASS, CLOB, COM-HANDLE, DATE, DATETIME, DATETIME-TZ, DECIMAL, HANDLE, INT64, INTEGER, LOGICAL, RAW, RECID, and ROWID.

For more information on these data types, see the Data types reference entry.

For the CLASS data type, you define a field in a temp-table as a class by specifying the built-in Progress.Lang.Object class name. For example:

```
DEFINE TEMP-TABLE ttObjHolder FIELD MyObj AS CLASS Progress.Lang.Object.
```

When you assign a class instance to a field, ABL implicitly casts the instance to its root super class, which is the Progress.Lang.Object class. After the assignment, the field contains an object reference to the class instance, not the object itself.

You cannot define a field in a database table as a class.

`LIKE` *`field`*

Specifies a database field or a variable whose characteristics the temp-table field inherits. If you name a variable with this option, that variable must have been defined earlier in the procedure. The temp-table field inherits the data type, extents, format, initial value, label, and column label.

If the database field is a COLUMN-CODEPAGE CLOB, the temp-table field is in the database field's code page. If the database field is a DBCODEPAGE CLOB, the temp-table field's code page is -cpinternal.

You can override selected characteristics of the field or variable with the *field-options* parameter.

If you reference a database field in the LIKE option, the database containing that field must be connected at both compile time and run time. Therefore, use the LIKE option with caution.

field-options

Specifies options for the temp-table field. Any options you specify override any options inherited through the LIKE option. This is the syntax for *field-options*:

```
{
   [ BGCOLOR expression ]
   [ COLUMN-LABEL label ]
   [ DCOLOR expression ]
   [ DECIMALS n ]
   [ EXTENT n ]
   [ FONT expression ]
   [ FGCOLOR expression ]
   [ FORMAT string ]
   [ HELP help-text ]
   [ INITIAL
     { constant | { [ constant [ , constant ] . . . ] } }
   ]
   [ LABEL label [ , label ] . . . ]
   [ MOUSE-POINTER expression ]
   [ [ NOT ] CASE-SENSITIVE ]
   [ PFCOLOR expression ]
   [ SERIALIZE-HIDDEN ]
   [ SERIALIZE-NAME serialize-name ]
   [ TTCODEPAGE | COLUMN-CODEPAGE codepage ]
   [ XML-DATA-TYPE string ]
   [ XML-NODE-TYPE string ]
   [ XML-NODE-NAME node-name ]
   { [ view-as-phrase ] }
}
```

HELP *help-text*

A quoted CHARACTER string that represents the help text.

SERIALIZE-HIDDEN

Indicates that this field is not written when the temp-table is serialized, for example into JSON or XML. This option also applies for serialization via the Progress.IO.JsonSerializer class.

Note: If you also specify XML-NODE-TYPE, the WRITE-XML( ) method uses the value of XML-NODE-TYPE and ignores this option.

SERIALIZE-NAME *serialize-name*

An optional CHARACTER constant that specifies the name of the temp-table field as it should appear when serialized, for example into JSON or XML. The default is temp-table *field-name*. This option is useful when:

- reading a temp-table field from XML or JSON with a name that contains invalid characters for an ABL name or is an ABL keyword, or
- writing a temp-table field to XML or JSON with a name that is an invalid XML tag name or an invalid JSON property name.

Note: If you set both XML-NODE-NAME and SERIALIZE-NAME, the READ-XML() and WRITE-XML() methods will use XML-NODE-NAME.

TTCODEPAGE | COLUMN-CODEPAGE *codepage*

Specifies the code page for a CLOB field in the temp-table. If you specify TTCODEPAGE, the code page is -cpinternal. If you specify COLUMN-CODEPAGE, *codepage* must be a valid code page name available in the DLC/convmap.cp file. You cannot specify the "undefined" code page for a CLOB. The code page you specify overrides any code page inherited through the LIKE option.

If you do not specify a code page for a CLOB field in the temp-table, the default code page is -cpinternal.

XML-DATA-TYPE *string*

An optional CHARACTER constant that specifies the XML Schema data type for the field in the temp-table. The XML Schema data type must be compatible with the ABL data type for the field.

For more information about the ABL XML data type mapping rules, see *Use XML with ABL Applications*.

XML-NODE-TYPE *string*

An optional CHARACTER constant that specifies the XML node type of the temp-table field, which lets you specify how the field is represented in XML. Valid option values are: "ATTRIBUTE", "ELEMENT", "HIDDEN", and "TEXT". The default value is "ELEMENT".

The following table lists the valid XML node types.

Table 34: XML node types

When the XML node type is . . .	The buffer field is . . .
ATTRIBUTE	Represented as an attribute of the temp-table element in both the XML Schema and data.
ELEMENT	Represented as a child element of the temp-table element in both the XML schema and data.

When the XML node type is . . .	The buffer field is . . .
HIDDEN	Omitted from both the XML Schema and data.
TEXT	Represented as a text element in both the XML Schema and data. **Note:** Each table can contain only one TEXT field. When a table contains a TEXT field, it cannot contain ELEMENT fields; it can contain only ATTRIBUTE fields. A table that contains a TEXT field cannot be part of a nested data-relation.

The XML node type of a temp-table field that represents an array must be either `"ELEMENT"` or `"HIDDEN"`.

Note: If you specify SERIALIZE-HIDDEN but do not set XML-NODE-TYPE, the AVM sets XML-NODE-TYPE to `"HIDDEN"`.

`XML-NODE-NAME` *`node-name`*

An optional CHARACTER constant that specifies the name of the XML element or XML attribute representing the temp-table field in an XML Document. The default is the temp-table *`field-name`*. This option is useful when:

- reading a temp-table field from XML with a name that contains invalid characters for an ABL name or is an ABL keyword, or
- writing a temp-table field to XML with a name that is an invalid XML tag name.

Note: If you set SERIALIZE-NAME but do not set XML-NODE-NAME, the AVM sets XML-NODE-NAME equal to SERIALIZE-NAME.

Note: You cannot specify an indeterminate array field in a temp-table using the EXTENT field option.

For more information and a description of all other field options, see the DEFINE VARIABLE statement.

`INDEX` *`index-name`*

`[[ AS | IS ][ UNIQUE ][ PRIMARY ][ WORD-INDEX ]]`

Defines an index on the temp-table. To define a unique index, specify the UNIQUE option. To define the primary index, specify the PRIMARY option. To define a word-index, specify the WORD-INDEX option.

If you define more than one index on the temp-table, you can specify PRIMARY for none or one of the indexes. If you specify PRIMARY for none of the indexes, the AVM makes the first index you specify the primary index.

If you define no indexes on the temp-table, and the temp-table does not inherit the indexes of another table through the LIKE option of the DEFINE TEMP-TABLE statement, the AVM creates a default index, makes it the primary index, and sorts the records in entry order. If you do not specify the USE-INDEX option under LIKE but do use the INDEX option, then no indexes are copied from the specified table unless the specified table is another temp-table.

index-field [ASCENDING | DESCENDING]

Specifies a temp-table field to use as a component of the index. You can use the ASCENDING or DESCENDING option to specify that the component has ascending or descending order.

If you do not specify a sort orientation (ASCENDING or DESCENDING), the index component gets the sort orientation of the previous index component, or, if there is no previous index component, ASCENDING. This rule applies only to index components of temp-tables.

Note: You cannot use a BLOB or CLOB field as a component of an index.

For example, the following two temp-table definitions are equivalent:

```
DEFINE TEMP-TABLE foo NO-UNDO
  FIELD a AS CHARACTER
  FIELD b AS CHARACTER
  FIELD c AS CHARACTER
  INDEX x a DESC b DESC c DESC.
```

```
DEFINE TEMP-TABLE foo NO-UNDO
  FIELD a AS CHARACTER
  FIELD b AS CHARACTER
  FIELD c AS CHARACTER
  INDEX x a DESC b c.
```

The following two temp-table definitions are also equivalent:

```
DEFINE TEMP-TABLE foo NO-UNDO
  FIELD a AS CHARACTER
  FIELD b AS CHARACTER
  FIELD c AS CHARACTER
  INDEX x a ASC b DESC c DESC.
```

```
DEFINE TEMP-TABLE foo NO-UNDO
  FIELD a AS CHARACTER
  FIELD b AS CHARACTER
  FIELD c AS CHARACTER
  INDEX x a ASC b DESC c.
```

Example

The following procedure creates a temp-table (`temp-item`) that stores the total inventory value (`Item.Price * Item.OnHand`) for each catalog page (`Item.CatPage`) in the `Sports2020` database. It builds temp-item with two indexes-one that sorts the table in ascending order by catalog page and a second that sorts the table in descending order by inventory value.

After building temp-item, the procedure displays a dialog box that prompts for report parameters. These parameters include the cutoff value of catalog page inventory to report, and whether to display the report by catalog page (ascending) or inventory value (descending). After displaying the report, the procedure displays another dialog box to repeat the process. The process is repeated until you press the CANCEL button. This procedure shows how you can use a temp-table to store a calculated result from the database, and efficiently report the same result according to different sorting and selection criteria:

r-tmptb1.p

```
DEFINE TEMP-TABLE temp-item
  FIELD cat-page  LIKE Item.CatPage
  FIELD inventory LIKE Item.Price LABEL "Inventory Value"
  INDEX cat-page  IS PRIMARY cat-page ASCENDING
  INDEX inventory-value inventory DESCENDING.
DEFINE VARIABLE cutoff      NO-UNDO LIKE item.price.
DEFINE VARIABLE inv-value   NO-UNDO LIKE item.price.
DEFINE VARIABLE report-type AS INTEGER NO-UNDO INITIAL 1.

DEFINE BUTTON ok-butt     LABEL "OK" AUTO-GO.
DEFINE BUTTON cancel-butt LABEL "CANCEL" AUTO-ENDKEY.
FORM
  cutoff LABEL "Inventory Lower Cutoff for each Catalog Page"
    AT ROW 1.25 COLUMN 2
  report-type LABEL "Report Sorted ..." AT ROW 2.25 COLUMN 2
    VIEW-AS RADIO-SET RADIO-BUTTONS
      "By Catalog Page",   1,
      "By Inventory Value", 2 SKIP
  ok-butt cancel-butt
  WITH FRAME select-frame SIDE-LABELS WIDTH 70
  TITLE "Specify Report ..." VIEW-AS DIALOG-BOX.

FOR EACH Item BREAK BY Item.CatPage:
  ACCUMULATE Item.Price * Item.OnHand (SUB-TOTAL BY Item.CatPage).
  IF LAST-OF(Item.CatPage) THEN DO:
    inv-value = ACCUM SUB-TOTAL BY Item.CatPage (Item.Price * Item.OnHand).
    CREATE temp-item.
    temp-item.cat-page = Item.CatPage.
    inventory = inv-value.
  END.
END. /* FOR EACH item */
ON CHOOSE OF ok-butt DO:
  HIDE FRAME select-frame.
  IF report-type = 1 THEN
    FOR EACH temp-item USE-INDEX cat-page WITH FRAME rpt1-frame:
      IF inventory >= cutoff THEN
        DISPLAY temp-item.cat-page inventory.
    END.
  ELSE
    FOR EACH temp-item USE-INDEX inventory-value WITH FRAME rpt2-frame:
      IF inventory >= cutoff THEN
        DISPLAY temp-item.cat-page inventory.
    END.
    VIEW FRAME select-frame.
END.

ENABLE ALL WITH FRAME select-frame.
WAIT-FOR CHOOSE OF cancel-butt OR WINDOW-CLOSE OF CURRENT-WINDOW.
```

For examples of instance and static temp-table data member definitions, see the descriptions of `r-CustObj.cls`, `r-CustObjStatic.cls`, and `r-CustObjAbstract.cls` in the CLASS statement reference entry.

Notes

- If you define a temp-table LIKE a database table, the temp-table does not inherit the database table's database triggers.
- You cannot define a temp-table field of type MEMPTR or LONGCHAR.
- You cannot define shared objects, work tables, or temp-tables within an internal procedure, a method in a class, or a user-defined function.
- A temp-table can be compile-time defined (often referred to as a *static* temp-table object), where the temp-table is defined and created at compile time using this statement, or it can be run-time defined (often referred to as a *dynamic* temp-table object), where the temp-table is defined and created at run time using the CREATE TEMP-TABLE statement and temp-table object handle operations. A compile-time defined temp-table can also be defined as a static data member of a class. In this case, it is a static temp-table object that is also a class static data member.
- ABL disregards the following options when used in conjunction with a temp-table:
  - The VALIDATE option on a DELETE statement.
  - The SHARE-LOCK, EXCLUSIVE-LOCK, and NO-LOCK options used with the FIND or FOR statements.
  - The NO-WAIT option on the FIND statement.
- Data handling statements that cause the AVM to automatically start a transaction for a regular table will not cause the AVM to automatically start a transaction for a temp-table. If you want to start a transaction for operations involving a temp-table, you must explicitly start a transaction by using the TRANSACTION keyword.
- Use the CASE-SENSITIVE option only when it is important to distinguish between uppercase and lowercase values entered for a character field. For example, use CASE SENSITIVE to define a field for a part number that contains mixed upper-case and lowercase characters.
- You cannot define a SHARED or NEW SHARED temp-table in a class definition (`.cls`) file. If you do, ABL generates a compilation error.
- A SHARED temp-table remains in scope for an instance of a persistent procedure until the instance is deleted. This is true even if the original procedure that defined the temp-table as NEW SHARED goes out of scope while the procedure instance remains persistent.

  If a trigger or internal procedure of a persistent procedure executes an external subprocedure that defines a SHARED temp-table, ABL includes the persistent procedure in the resolution of the corresponding NEW SHARED temp-table as though the procedure were on the procedure call stack.
- You can specify a join between a temp-table or work table and any appropriate table using the OF keyword. The two tables must contain a commonly named field that participates in a unique index for at least one of the tables. For more information on table joins see the Record phrase reference entry.
- If you define a temp-table with the same name as a database table and then you define a buffer for that name, the buffer will be associated with the database table, not with the temp-table.
- The point at which the AVM stores temp-table overflow from memory to disk is based on the setting of the Number of Buffers for Temp-tables (`-Bt`) startup parameter, which specifies the number of buffers in the temp-table database pool. For more information, see *Startup Command and Parameter Reference*.
- See *Develop ABL Applications* for information on temp-tables and work tables.

See also

Class-based data member access, CREATE-LIKE( ) method, CREATE-LIKE-SEQUENTIAL( ) method, CREATE TEMP-TABLE statement, DEFINE DATASET statement, DEFINE WORK-TABLE statement, NUM-REFERENCES attribute, RUN statement

DEFINE VARIABLE statement

Defines a variable for use in one or more procedures, a variable data member of a class for use in a single class or class hierarchy, or by other classes and procedures, or a variable data element for use within a single class-based method.

The VAR statement is a shorthand notation for the DEFINE VARIABLE statement and may be used for NO-UNDO variables. For more information, see the VAR statement in the *ABL Reference*.

Syntax

```
DEFINE {[[ NEW [ GLOBAL ]] SHARED ]|
          [ PRIVATE | PACKAGE-PRIVATE | PROTECTED | PACKAGE-PROTECTED | PUBLIC ]
          [ STATIC ][ SERIALIZABLE | NON-SERIALIZABLE ]}
  VARIABLE variable-name
  {{   AS primitive-type-name
     | AS [ CLASS ]{object-type-name}
     | LIKE field       }[ EXTENT [constant]]}
  [ SERIALIZE-NAME serialize-name ]
  [ BGCOLOR expression]
  [ COLUMN-LABEL label]
  [ CONTEXT-HELP-ID expression]
  [ DCOLOR expression]
  [ DECIMALS n]
  [ DROP-TARGET ]
  [ FONT expression]
  [ FGCOLOR expression]
  [ FORMAT string]
  [ INITIAL
    { constant |{[ constant[ , constant]...]}}]
  [ LABEL string[ , string]...]
  [ MOUSE-POINTER expression]
  [ NO-UNDO ]
  [[ NOT ] CASE-SENSITIVE ]
  [ PFCOLOR expression]
  {[ view-as-phrase ]}
  {[ trigger-phrase ]}
```

NEW SHARED VARIABLE *variable-name*

Defines and identifies a variable to be shared by a procedure called directly or indirectly by the current procedure. The called procedure must name the same variable in a DEFINE SHARED VARIABLE statement.

`NEW GLOBAL SHARED VARIABLE` *`variable-name`*

Defines and identifies a variable that can be used by any procedure that names that variable using the DEFINE SHARED VARIABLE statement. The value of a global shared variable remains available throughout an ABL session.

`SHARED VARIABLE` *`variable-name`*

Defines and identifies a variable that was created by another procedure that used the DEFINE NEW SHARED VARIABLE or DEFINE NEW GLOBAL SHARED VARIABLE statement.

`[ PRIVATE | PACKAGE-PRIVATE | PROTECTED | PACKAGE-PROTECTED | PUBLIC ] [ STATIC ] [ SERIALIZABLE | NON-SERIALIZABLE ] VARIABLE` *`variable-name`*

Defines and identifies a variable as a data member of a class, and optionally specifies an access mode (PRIVATE, PACKAGE-PRIVATE, PROTECTED, PACKAGE-PROTECTED, or PUBLIC) and scope (instance or STATIC) for that data member, as well as whether or not the variable will participate in serialization (SERIALIZABLE or NON-SERIALIZABLE).

You cannot specify these options when defining a variable as a data element of a method (including constructors, destructors, and property accessors) or procedure.

Note: The specified options are applicable only when defining a data member for a class in a class definition (`.cls`) file. The access mode modifier and the STATIC modifier can appear in either order, e.g., `DEFINE STATIC PUBLIC VARIABLE myVariable...` is valid syntax. Note also that you cannot override the definition of a given variable data member in a class hierarchy.

PRIVATE variable data members can be accessed by the defining class. An instance can access a private data member of another instance if both instances are from the same class. PACKAGE-PRIVATE variable data members can be accessed from within the class and any class within its package. PROTECTED variable data members can be accessed by the defining class and any of its derived classes. PACKAGE-PROTECTED variable data members can be accessed from within the class, any class within its package, and from within any subclass that inherits the class. PUBLIC variable data members can be accessed from any code (in a class or a procedure) that has access to a class instance.

The default access mode is PRIVATE.

A variable defined with the STATIC option is a static data member of the class type for which it is defined and is scoped to the ABL session where it is referenced. ABL creates one copy of the specified class static variable on first reference to the class type, and ABL creates only one such copy for any number of instances of the class that you create. You can reference an accessible static variable data member in any piece of code.

Without the STATIC option, ABL creates an instance variable data member that is scoped to a single instance of the class where it is defined. ABL creates one copy of the specified instance variable for each such class instance that you create. You can reference any public instance variable in any procedure, or in any instance or static method defined inside or outside of the class where the instance variable is defined. Any static method can reference the public instance variable only using an object reference to a class instance that defines the variable as a data member. If the referencing static method is defined in the same class as the public instance variable, the class must instantiate itself in order to have access to an instance reference.

For more information on the mechanism for accessing variable data members of different access modes and scopes, see the reference entry for Class-based data member access.

Note: Members of a class are grouped into six namespaces, including buffers/temp-tables, methods, variables/properties/events, ProDataSets, queries, and data-sources. Variables, properties, and events defined as members of a class share the same namespace. There can be only one class member in this namespace with a given name (not counting abstract member overrides).

Class-based objects that are defined as serializable (using the SERIALIZABLE option in the CLASS statement) can be passed as parameters in remotes call between the application server and ABL clients and can be serialized to binary or JSON format. By default, both passing a class as a parameter and serializing an object to binary via the `Progress.IO.BinarySerializer` class include all data members regardless of access mode. However, for JSON serialization via `Progress.IO.JsonSerializer`, only public data members are serialized. To include other access modes during JSON serialization, SERIALIZABLE must be added to the definition. See Serialize( ) method (JsonSerializer) for more information.

Use the NON-SERIALIZABLE option to exclude a given variable from the serialization process via the `Progress.IO.BinarySerializer` or `Progress.IO.JsonSerializer` class. A variable marked as NON-SERIALIZABLE is assigned its initial value when the class is deserialized.

You can also use the NON-SERIALIZABLE option to make a class that otherwise would not be serializable because one or more of its members. (See the Parameter passing syntax entry for full details on what can and cannot be serialized.) For example, if a variable itself is a class-based object that is not serializable, adding the NON-SERIALIZABLE keyword to that variable will allow the class that contains it to be defined as SERIALIZABLE.

Note: Use the SERIALIZABLE and NON-SERIALIZABLE options only with instance variables. These options are not for use with variables defined as STATIC or as part of an interface.

If *`variable-name`* is an ABL reserved keyword, you must include the access mode in its definition. You must also use the THIS-OBJECT system reference or Type-name syntax to access the variable data member. For more information on referencing variable data members, see the reference entry for Class-based data member access.

Note: If the data member is defined in a class that is derived from a .NET class, other restrictions on *`variable-name`* apply. For more information, see the CLASS statement reference entry.

For more information on where and how to define data members in a class, see the CLASS statement reference entry.

`VARIABLE` *`variable-name`*

Defines and identifies a variable whose value you can access only within the current procedure, method of a class (including constructors, destructors, and property accessors), or as a PRIVATE data member of a class.

`AS` *`primitive-type-name`*

Specifies a built-in primitive type for the variable you are defining. The built-in data type (*`primitive-type-name`*) can be one of the following:

```
CHARACTER | COM-HANDLE | DATE | DATETIME | DATETIME-TZ | DECIMAL
| HANDLE | INT64 | INTEGER | LOGICAL | LONGCHAR | MEMPTR | RAW
| RECID | ROWID
```

For more information on these primitive types, see the Data types reference entry.

AS [CLASS] { *object-type-name* }

Defines the variable as an object reference with the data type of a class or interface. The default value of the variable is the Unknown value (?). You cannot assign an initial value using the INITIAL option.

object-type-name

Specifies the type name of an ABL or .NET class or interface. Specify an object type name using the syntax described in the Type-name syntax reference entry. With an appropriate USING statement, you can also specify a class or interface name alone, without the qualifying package or namespace.

You cannot directly specify the type name of a .NET mapped object type (such as System.Int32). To define a variable that matches a .NET mapped type, you must define it as the corresponding ABL primitive type (*primitive-type-name*).

CLASS

If the specified class or interface type name conflicts with an abbreviation for a built-in primitive type name, such as INT for INTEGER, you must specify the CLASS keyword.

For more information on object references, see the Class-based object reference reference entry.

LIKE *field*

Indicates the name of the variable, database field, temp-table field, or work table field whose characteristics you want to use for the variable you are defining. If you name a variable with this option, you must have defined that variable earlier in the procedure. You can override the format, label, initial value, decimals, and extent of the variable or database field by using the FORMAT, LABEL, COLUMN-LABEL, INITIAL, DECIMALS, EXTENT, and VIEW-AS options. If you do not use these options, the variable takes on the characteristics of the variable or database field you name.

If *field* has help and validate options defined, the variable you are defining does **not** inherit those characteristics.

If you reference a database field in a LIKE option in a DEFINE VARIABLE statement, DEFINE TEMP-TABLE statement, DEFINE WORK-TABLE statement, or format phrase, the database containing the referenced field must be connected at compile time but not necessarily at run time. Therefore, use the LIKE option with caution.

EXTENT [*constant*]

Defines the variable as an array of data elements, where the element data type is specified by the AS *primitive-type-name* option, the LIKE *field* option, or the AS *object-type-name* option. This option can specify an array variable as either determinate (has a defined number of elements) or indeterminate (has an undefined number of elements). To define a determinate array variable, specify the EXTENT option with the *constant* argument. This optional argument is an integer value that represents the number of data elements in the array variable. To define an indeterminate array variable, specify the EXTENT option without the *constant* argument.

The EXTENT is part of the variable data type. For more information, see the Type-name syntax reference entry.

An indeterminate array variable can be in one of two states: with a size (fixed) or without a size (unfixed). An indeterminate array variable does not have a size when first defined, unless initial values are provided.. You can set the size of an indeterminate array variable by:

- Initializing the array values when you define the variable, using the INITIAL option
- Setting the number of elements in the array variable using the EXTENT statement
- Assigning a sized array to the indeterminate array, setting the size to the size of the source array
- Passing array parameters to a procedure, user-defined function, or class-based method, so that the indeterminate array variable is the target for the passing of a determinate array, setting the size to the size of the determinate array

If you want to define a variable that is like an array variable or field, using the LIKE option, but if you do not want the variable to be an array, you can use EXTENT 0 to indicate a non-array field.

If you are using the AS option and you do not use the EXTENT option (or you specify *constant* as 0), the variable is not an array variable. If you are using the LIKE *field* option and you do not use the EXTENT option, the variable uses the extent defined for the database field you name (if any).

`SERIALIZE-NAME` *serialize-name*

An optional CHARACTER constant that specifies the name of the variable as it should appear when serialized into JSON. The default is *variable-name*. Use this option when the serialized name contains invalid characters for an ABL name, is an ABL keyword, or an invalid JSON property name.

This option is only valid for class members, and it cannot be used if the variable is defined as NON-SERIALIZABLE or the class that contains it is not defined as SERIALIZABLE.

`BGCOLOR` *expression*

Specifies a background color for the variable in graphical interfaces. This option is ignored in character interfaces.

[`NOT` **]** `CASE-SENSITIVE`

CASE-SENSITIVE indicates that the value stored for a character variable is case sensitive, and that all comparisons operations involving the variable are case sensitive. If you do not use this option, ABL comparisons are usually case insensitive. If you define a variable LIKE another field of variable, the new variable inherits case sensitivity. Use [NOT] CASE-SENSITIVE to override this default.

`COLUMN-LABEL` *label*

Names the label you want to display above the variable data in a frame that uses column labels. If you want the label to use more than one line (a stacked label), use an exclamation point (!) in the label to indicate where to break the line. For example:

r-collbl.p

```
DEFINE VARIABLE credit-percent AS INTEGER NO-UNDO
  COLUMN-LABEL "Enter   !percentage!increase ".

FOR EACH Customer:
  DISPLAY Customer.Name Customer.CreditLimit.
  SET credit-percent.
  Customer.CreditLimit = (Customer.CreditLimit *
    (credit-percent / 100)) + Customer.CreditLimit.
  DISPLAY Customer.CreditLimit @ new-credit LIKE Customer.CreditLimit
    LABEL "New max cred".
END.
```

If you want to use the exclamation point (!) as one of the characters in a column label, use two exclamation points (`!!`).

The AVM does not display column labels if you use the SIDE-LABELS or NO-LABELS options with the Frame phrase.

If you define a variable to be LIKE a field, and that field has a column label in the Data Dictionary, the variable inherits that column label.

`CONTEXT-HELP-ID` *`expression`*

An integer value that specifies the identifier of the help topic for this variable in a help file specified at the session, window or dialog box level using the CONTEXT-HELP-FILE attribute.

`DCOLOR` *`expression`*

Specifies the display color for the variable in character interfaces. This option is ignored in graphical interfaces.

`DECIMALS` *`n`*

Specifies the number of decimal places to store for a DECIMAL variable, where *n* is an integer constant. When you define a variable AS DECIMAL, ABL automatically stores up to 10 decimal places for the value of that variable. Use the DECIMALS option to store a smaller number of decimal places. The DECIMALS option has nothing to do with the display format of the variable, just the storage format.

If you use the LIKE option to name a field whose definition you want to use to define a variable, ABL uses the number of decimals in the field definition to determine how many decimal places to store for the variable.

Note: The XML-WRITE() method on a temp-table, temp-table buffer, or ProDataSet object also uses the DECIMALS attribute to determine the number of decimal places to export to the XML Document.

`DROP-TARGET`

Indicates whether you want to be able to drop a file onto the object.

The following example shows setting the DROP-TARGET option for a variable:

```
DEFINE VARIABLE fill-in-1 AS CHARACTER DROP-TARGET.
```

`FGCOLOR` *`expression`*

Specifies a foreground color for the variable in graphical interfaces. This option is ignored in character interfaces.

`FONT` *`expression`*

Specifies a font for the variable.

`FORMAT` *`string`*

The data format of the variable you define. If you use the AS *`primitive-type-name`* option and you do not use FORMAT *`string`*, the variable uses the default format for its data type. The following table lists the default data formats for the data types.

Table 35: Default display formats

Data type	Default display format
BLOB	(See note)
CHARACTER	x(8)
CLASS	N/A (See note)
CLOB	(See note)
COM-HANDLE	>>>>>>9
DATE	99/99/99
DATETIME	99/99/9999 HH:MM:SS.SSS
DATETIME-TZ	99/99/9999 HH:MM:SS.SSS+HH:MM
DECIMAL	->>,>>9.99
HANDLE	>>>>>>9
INT64	->,>>>,>>9
INTEGER	->,>>>,>>9
LOGICAL	yes/no
LONGCHAR	(See note)
MEMPTR	(See note)
RAW	(See note)
RECID	>>>>>>9
ROWID	(See note)

Note: You cannot display a BLOB, CLOB, MEMPTR, RAW, or ROWID value directly. However, you can convert a MEMPTR, RAW, or ROWID value to a character string representation using the STRING function and display the result. You can also convert a BLOB to a MEMPTR, and then use the STRING function. A MEMPTR or RAW value converts to decimal integer string. A ROWID value converts to a hexadecimal string, "0x*hexdigits*", where *hexdigits* is any number of characters "0" through "9" and "A" through "F". You can only display a LONGCHAR using the VIEW-AS EDITOR LARGE phrase. You can display a CLOB field by converting it to a LONGCHAR.

Note: If you display a class instance using the MESSAGE statement, ABL automatically invokes the ToString( ) method (Object) (provided by the Progress.Lang.Object class) on the object reference. To display a class instance in a frame (for example, using the DISPLAY statement), you must first explicitly convert the object reference to a displayable type using the INT64 function, the INTEGER function, the STRING function, or the ToString( ) method (Object).

See *Develop ABL Applications* for more information on data formatting.

If you use the LIKE *field* option and you do not use the FORMAT *string* option, the variable uses the format defined for the database field you name. You must enclose the *string* in quotes.

INITIAL { *constant* | [*constant* [, *constant*] . . .] }

The initial value of the variable you want to define. If you use the AS *primitive-type-name* option and you do not use the INITIAL *constant* option, the default is the initial value for the data type of the variable or array element.

When you define an array variable, you can supply initial values for each element in the array. For example:

```
DEFINE VARIABLE array-var AS CHARACTER NO-UNDO EXTENT 3
  INITIAL ["Add","Delete","Update"].
```

If you do not supply enough values to fill up the elements of the array, the AVM puts the last value you named into the remaining elements of the array. If you supply too many values, the AVM raises an error.

If you define a variable as an indeterminate array, and you supply initial values for elements in the array, the AVM sets the size to the number of elements in the array. You can then use the array as you would a determinate array, but you can also change its size later, if desired.

You can also use the EXTENT statement to set or reset the number of elements in an indeterminate array variable. For more information, see the EXTENT statement reference entry.

The following table lists the default initial values for the various variable data types.

Table 36: Default variable initial values

Data type	Default initial value
CHARACTER	"" (an empty string)
CLASS	Unknown value (?) (See note)

Data type	Default initial value
COM-HANDLE	Unknown value (?) (See note)
DATE	Unknown value (?) (displays as blanks)
DATETIME	Unknown value (?)
DATETIME-TZ	Unknown value (?)
DECIMAL	0
HANDLE	Unknown value (?) (See note)
INT64	0
INTEGER	0
LOGICAL	no
LONGCHAR	Unknown value (?)
MEMPTR	A zero-length sequence of bytes (See note)
RAW	A zero-length sequence of bytes (See note)
RECID	Unknown value (?)
ROWID	Unknown value (?) (See note)

Note: You cannot use the INITIAL option of the variable definition to specify an initial value for CLASS, COM-HANDLE, HANDLE, MEMPTR, RAW, and ROWID data types.

If you are using the LIKE *field* option and you do not use the INITIAL *constant* option, the variable uses the initial value of the field or variable. In the DEFINE SHARED VARIABLE statement, the INITIAL option has no effect. However, the DEFINE NEW SHARED VARIABLE, the DEFINE NEW SHARED TEMP-TABLE, and the DEFINE NEW WORK-TABLE statements work with the INITIAL option.

LABEL *string* **[** *, string* **] . . .**

The label you want to use when the variable is displayed. If you use the AS *primitive-type-name* option and you do not use the LABEL *string* option, the default label is the variable name. If you

use the LIKE *field* option and you do not use the LABEL *string* option, the variable uses the label of the field or variable you name. You must enclose the *string* in quotes.

You can specify a label for each element in a determinate array variable. You cannot specify a label for elements in an indeterminate array variable.

In MS-Windows, you can designate a character within each label as a navigation mnemonic. Precede the character with an ampersand (&). When the variable is displayed with side labels, the mnemonic is underlined. The user can move focus to the variable by pressing **ALT** and the underlined letter. Navigation mnemonics operate only when you use side labels. If you specify more than one widget with the same mnemonic, the AVM transfers focus to each of these in tab order when you make a selection.

Ending a label with an ampersand might produce unwanted behavior. To include a literal ampersand within a label, specify a double ampersand (&&).

MOUSE-POINTER *expression*

Specifies the default mouse pointer for the variable.

NO-UNDO

When the value of a variable is changed during a transaction and the transaction is undone, the AVM restores the value of the variable to its prior value. If you do not want, or if you do not need, the value of a variable to be undone even when it has been changed during a transaction, use the NO-UNDO option with the DEFINE VARIABLE statement. NO-UNDO variables are more efficient; use this option whenever possible.

Specifying NO-UNDO for a variable is especially useful if you want to indicate an error condition as the value of the variable, perform an UNDO, and later take some action based on that error condition. If one variable is defined LIKE another that is NO-UNDO, the second variable will be NO-UNDO only if you specify NO-UNDO in the definition of the second variable.

PFCOLOR *expression*

Specifies the prompt-for color for the variable in character interfaces. This option is ignored in graphical interfaces.

view-as-phrase

Specifies the default data representation widget for this variable. Following is the syntax for the *view-as-phrase*:

```
VIEW-AS
  {     combo-box-phrase
      | editor-phrase
      | FILL-IN
          [ NATIVE ]
          [size-phrase]
          [ TOOLTIP tooltip]
      | radio-set-phrase
      | selection-list-phrase
      | slider-phrase
      | TEXT
          [size-phrase]
          [ TOOLTIP tooltip]
      | TOGGLE-BOX
          [size-phrase]
          [ TOOLTIP tooltip]
  }
```

For more information on *view-as-phrase*, see the VIEW-AS phrase reference entry.

trigger-phrase

Defines triggers for the data representation widget specified in the *view-as-phrase*. Following is the syntax for the *trigger-phrase*:

```
TRIGGERS:
  { ON event-list[ ANYWHERE ]
      {     trigger-block
          | PERSISTENT RUN proc-name
              [ IN handle]
              [ ( input-parameters ) ]
      }
  }...
END [ TRIGGERS ]
```

For more information on triggers, see the Trigger phrase reference entry.

Example

The `r-dfvar.p` procedure defines two variables, `del` and `nrecs` to be shared with procedure `r-dfvar2.p`. The `del` variable passes information to `r-dfvar2.p`, while `nrecs` passes information back to `r-dfvar.p` from `r-dfvar2.p`.

r-dfvar.p

```
DEFINE NEW SHARED VARIABLE del   AS LOGICAL NO-UNDO.
DEFINE NEW SHARED VARIABLE nrecs AS INTEGER NO-UNDO.

MESSAGE "Do you want to delete the orders being printed (y/n)?" UPDATE del.
RUN r-dfvar2.p.
IF del THEN
  MESSAGE nrecs "orders have been shipped and were deleted".
ELSE
  MESSAGE nrecs "orders have been shipped".
```

r-dfvar2.p

```
DEFINE SHARED VARIABLE del   AS LOGICAL NO-UNDO.
DEFINE SHARED VARIABLE nrecs AS INTEGER NO-UNDO.

OUTPUT TO PRINTER.
FOR EACH Order WHERE Order.ShipDate <> ?:
   nrecs = nrecs + 1.
   FOR EACH OrderLine OF Order:
     DISPLAY OrderLine.OrderNum OrderLine.LineNum OrderLine.Qty
       OrderLine.Price.
       IF del THEN DELETE OrderLine.
   END.
   IF del THEN DELETE Order.
END.
OUTPUT CLOSE.
```

The following example is a startup procedure. It defines a new global variable with the initial value TRUE and uses that variable to determine whether to run an initialization procedure, `r-init.p`, that displays sign-on messages. Then the global variable first-time is set to FALSE. If you restart this procedure during the same session (pressed **STOP**), `r-init.p` does not run again.

The procedure also defines the variable selection for entering menu choices within this procedure:

r-dfvar3.p

```
DEFINE NEW GLOBAL SHARED VARIABLE first-time AS LOGICAL NO-UNDO INITIAL TRUE.
DEFINE VARIABLE selection AS INTEGER NO-UNDO FORMAT "9" LABEL "Selection".

IF first-time THEN DO:
  RUN r-init.p.
  first-time = FALSE.
END.

FORM
  "    MAIN MENU          " SKIP(1)
  "1 - Accounts Payable   " SKIP
  "2 - Accounts Receivable"
  WITH CENTERED ROW 5 FRAME menu.

REPEAT:
  VIEW FRAME menu.
  UPDATE selection AUTO-RETURN WITH FRAME sel CENTERED ROW 12 SIDE-LABELS.
  IF selection = 1 THEN DO:
    HIDE FRAME menu.
    HIDE FRAME sel.
    RUN apmenu.p.
  END.
  ELSE IF selection = 2 THEN DO:
    HIDE FRAME menu.
    HIDE FRAME sel.
    RUN armenu.p.
  END.
  ELSE DO:
    MESSAGE "Invalid selection. Try again".
    UNDO, RETRY.
  END.
END.
```

The following procedure finds the day of the week of a date the user enters. The procedure defines an array with seven elements and uses the INITIAL option to define the initial value of each element in the array.

r-dfvar4.p

```
DEFINE VARIABLE dow AS CHARACTER NO-UNDO FORMAT "x(9)" EXTENT 7
  INITIAL ["Sunday", "Monday", "Tuesday", "Wednesday",
  "Thursday", "Friday", "Saturday"].

DEFINE VARIABLE dob AS DATE NO-UNDO INITIAL TODAY.

REPEAT WITH SIDE-LABELS 1 DOWN CENTERED ROW 10 TITLE "Date of Birth":
  DISPLAY SKIP(1).
  UPDATE dob LABEL "Enter date of birth".
  DISPLAY dow[WEEKDAY(dob)] LABEL "It was a".
END.
```

The following example defines a variable with a VIEW-AS phrase and a Trigger phrase:

r-defsel.p

```
DEFINE VARIABLE ix    AS INTEGER   NO-UNDO.
DEFINE VARIABLE clubs AS CHARACTER NO-UNDO
  VIEW-AS SELECTION-LIST SIZE 20 BY 5 MULTIPLE SCROLLBAR-VERTICAL NO-DRAG
    LIST-ITEMS "One Iron", "Two Iron", "Three Iron", "Four Iron",
               "Five Iron", "Six Iron", "Seven Iron", "Eight Iron",
               "Nine Iron", "Pitching Wedge"
  LABEL "Golf Clubs Available"
  TRIGGERS:
    ON GO DO:
      IF SELF:SCREEN-VALUE <> "" THEN
      DO ix = 1 TO NUM-ENTRIES(SELF:SCREEN-VALUE) :
        DISPLAY ENTRY(ix, SELF:SCREEN-VALUE) FORMAT "X(16)"
          WITH FRAME clubs-sel CENTERED
            NUM-ENTRIES(SELF:SCREEN-VALUE) + 1 DOWN
            TITLE "Clubs Selected" USE-TEXT.
        DOWN 1 WITH FRAME clubs-sel.
      END.
    END.
  END TRIGGERS.

ENABLE clubs WITH FRAME get-info TITLE "Select the Desired Club(s)".
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

For examples of instance and static variable data member definitions, see the descriptions of `r-CustObj.cls`, `r-CustObjStatic.cls`, and `r-CustObjAbstract.cls` in the CLASS statement reference entry.

Notes

- You can use the DEFINE VARIABLE statement anywhere. However, all references to the variable must appear after the DEFINE VARIABLE statement that defines it.
- You cannot define a variable as a BLOB or CLOB field. You can define a variable using their MEMPTR and LONGCHAR counterparts, respectively.
- A HANDLE variable defined as a static data member can reference the handle of any compile-time defined (static) or run-time defined (dynamic) handle-based object, such as a button, temp-table, or ProDataSet.
- Defining a LONGCHAR variable supports the same options as a CHARACTER variable, except for the FORMAT option and all VIEW-AS options except VIEW-AS EDITOR LARGE.
- You should use the CASE-SENSITIVE option only when it is important to distinguish between uppercase and lowercase values entered for a character variable. For example, use CASE-SENSITIVE to define a variable for a part number that contains mixed uppercase and lowercase characters.
- After you use the DEFINE NEW GLOBAL SHARED VARIABLE statement to create a global shared variable, use the DEFINE SHARED VARIABLE statements in other procedures to access that variable.
- You cannot define the same global variable twice in the same ABL session. If you try, and the definitions of the two variables do not match, the AVM raises an error. If the definitions of the two variables match, the AVM disregards the second variable you tried to define (if you are rerunning a startup procedure).
- Changes made to variables when there is no active transaction are not undone when a block is undone.
- When a procedure names and uses a shared variable:
  - The AVM searches through the calling chain of procedures looking for the most recent DEFINE NEW SHARED VARIABLE statement that created that shared variable.
  - If no DEFINE NEW SHARED VARIABLE statement is found, the AVM searches for a DEFINE NEW GLOBAL SHARED VARIABLE statement that created the shared variable.

  - If the procedure that names the shared variable is called from a trigger or internal procedure that is part of a persistent procedure context, the persistent context is also checked for the most recent DEFINE NEW SHARED VARIABLE or DEFINE NEW GLOBAL SHARED VARIABLE statement at the point in the calling chain where the trigger or internal procedure is executed.
  - If the AVM finds one of these statements, it does not search any further for other statements that might have defined the same variable as NEW or NEW GLOBAL.

- The AVM checks the definition of a SHARED variable against that of the corresponding NEW SHARED or NEW GLOBAL SHARED variable. The data types and array extents must match. If the FORMAT, LABEL and DECIMALS specifications are not the same, each procedure uses its individual specification. The DEFINE NEW SHARED statement determines if a shared variable is NO-UNDO.
- A SHARED variable remains in scope for an instance of a persistent procedure until the instance is deleted. This is true even if the original procedure that defined the variable as NEW SHARED goes out of scope while the procedure instance remains persistent.

  If a trigger or internal procedure of a persistent procedure executes an external subprocedure that defines a SHARED variable, the AVM includes the persistent procedure in the resolution of the corresponding NEW SHARED variable as though the procedure were on the procedure call stack.
- If an application with several procedures defines a NEW SHARED variable with the same name in each procedure, the AVM creates a different instance of the NEW SHARED variable in each procedure. This behavior supports recursive procedures and bill-of-materials applications.
- You can neither define a SHARED or NEW SHARED variable, nor access such a variable defined in a procedure file, from within a class definition (`.cls`) file. If you do, ABL generates a compilation error. However, multiple procedure (`.p`) files can define and access an object reference variable for a class instance as a NEW SHARED or NEW GLOBAL SHARED variable. In this case, the object reference variables must be defined for the same class (not a subclass or a super class) in all procedures that use them.
- For SpeedScript, the following options are invalid: BGCOLOR, CONTEXT-HELP-ID, DCOLOR, FONT, FGCOLOR, MOUSE-POINTER, PFCOLOR, and *view-as-phrase*.

See also

Class-based data member access, DEFINE BUFFER statement, RUN statement, Trigger phrase, Type-name syntax, USING statement, VAR statement, VIEW-AS phrase

---

DEFINE WORK-TABLE statement

Defines a work table (a temp-table stored in memory) for use in one or more procedures, or within a single class.

This statement is supported only for backward compatibility. Use the DEFINE TEMP-TABLE statement, instead.

Syntax

```
DEFINE { [ [ NEW ] SHARED ] | [ PRIVATE ] }
  { WORK-TABLE | WORKFILE } work-table-name [ NO-UNDO ]
  [ LIKE tablename [ VALIDATE ] ]
  [ FIELD field-name { AS data-type | LIKE field }
    [ field-options ] ] . . .
```

NEW SHARED { WORK-TABLE | WORKFILE } *work-table-name*

Defines and identifies a work table to be shared by a procedure called directly or indirectly by the current procedure. The called procedure must name the same work table in a DEFINE SHARED WORK-TABLE statement. The WORKFILE keyword is allowed only for backward compatibility; using WORK-TABLE or WORKFILE has the same effect.

SHARED { WORK-TABLE | WORKFILE } *work-table-name*

Defines and identifies a work table that was defined by another procedure that used the DEFINE NEW SHARED WORK-TABLE statement. The WORKFILE keyword is allowed only for backward compatibility; using WORK-TABLE or WORKFILE has the same effect.

[PRIVATE] { WORK-TABLE | WORKFILE } *work-table-name*

Defines and identifies a work-table as a class-scoped object. A class-scoped handle-based object is not a member of a class, but provides a resource that is privately available to the class definition similar to a non-shared data element in a procedure definition. The option to specify the PRIVATE access mode is provided for readability. You cannot specify PRIVATE when defining a work-table as a data element in a method or procedure. The WORKFILE keyword is allowed only for backward compatibility; using WORK-TABLE or WORKFILE has the same effect.

Note: This option is applicable only when defining a class-scoped work-table in a class definition (`.cls`) file.

{ WORK-TABLE | WORKFILE } *work-table-name*

Defines and identifies a work table whose records you can access only within the current procedure, class, or method of a class.

The WORKFILE keyword is allowed only for backward compatibility; using WORK-TABLE or WORKFILE has the same effect.

NO-UNDO

Specifies that the AVM should not restore the record to its prior condition when a work table record is changed during a transaction and the transaction is undone. If you do not want the work table record undone even if it has changed during a transaction, use the NO-UNDO option with the DEFINE WORK-TABLE statement. NO-UNDO work tables are more efficient; use them whenever possible.

LIKE *table-name*

Indicates the name of a table whose characteristics you want to use for the work table you are defining. All of the fields in this base table are also in the work table. If you reference a database table with the LIKE option, the database containing that table must be connected at compile time. It need not be connected at run time.

If more than one connected database contains a table named *table-name*, you must qualify the table name with the database name. See the Record phrase description for more information.

HELP options are inherited from the *table-name*. Validate options are inherited only if the VALIDATE keyword is used.

VALIDATE

The work table fields inherit, from the dictionary, validation expressions and validation messages from the database table, *table-name*.

FIELD *field-name*

Identifies the name of a field in the work table.

AS *data-type*

Indicates the data type of the field or variable you are defining. The data types are CHARACTER, COM-HANDLE, DATE, DATETIME, DATETIME-TZ, DECIMAL, HANDLE, INT64, INTEGER, LOGICAL, RAW, RECID, and ROWID.

For more information on these data types, see the Data types reference entry.

LIKE *field*

Indicates the name of the variable, database field, temp-table field, or work table field whose characteristics you want to use for the work table field you are defining. If you name a variable with this option, you must have defined that variable earlier in the procedure. The work table field inherits the data type, extents, format, initial value, label, and column label of the *field*. You can override specific values by using the FORMAT, LABEL, INITIAL, DECIMALS, and EXTENT options. If you do not use these options, the field or variable takes on the characteristics of the variable or database field you name.

If you reference a database field in the LIKE option, the database containing that field must be connected at both compile time and run time. Therefore, use the LIKE option with caution.

field-options

Specifies options for the temp-table field. Any options you specify override any options inherited through the LIKE option. This is the syntax for *field-options*:

```
{  [ BGCOLOR expression ]
   [ COLUMN-LABEL label ]
   [ DCOLOR expression ]
   [ DECIMALS n ]
   [ EXTENT n ]
   [ FONT expression ]
   [ FGCOLOR expression ]
   [ FORMAT string ]
   [ INITIAL
     { constant | { [ constant [ , constant ] . . . ] } } ]
   [ LABEL label [ , label ] . . . ]
   [ MOUSE-POINTER expression ]
   [ [ NOT ] CASE-SENSITIVE ]
   [ PFCOLOR expression ]
   { [ view-as-phrase ] } }
```

Note: You cannot specify a BLOB field, a CLOB field, or an indeterminate array field in a work-table.

For a description of each option, see the DEFINE VARIABLE statement.

Example

The `r-wrkfil.p` procedure accumulates all balances by state and stores that information for display later. The procedure uses a work table to accomplish this task.

The `r-wrkfil.p` procedure defines the work table showsales. The work table contains the three fields named region, state, and tot-sales. These fields have all the same characteristics (except labels) as the Customer.SalesRegion, Customer.State, and Customer.Balance fields, respectively.

The first FOR EACH loop in the `r-wrkfil.p` procedure sorts Customers by state. Then it accumulates the balances for each Customer by state. When the procedure finds the last Customer in a state, it creates a showsales record for that state. The procedure assigns information to the fields in the showsales record. After looking at each Customer, the procedure continues to the next FOR EACH statement.

The second FOR EACH statement in the `r-wrkfil.p` procedure uses the information stored in the showsales table. Because you treat a work table within a procedure the same way you treat a database table, you can perform the same work with the showsales table that you can with a database table.

r-wrkfil.p

```
DEFINE WORK-TABLE showsales
  FIELD region    LIKE SaleRrep.Region LABEL "Region"
  FIELD state     LIKE Customer.State LABEL "St"
  FIELD tot-sales LIKE Customer.Balance COLUMN-LABEL "Total!Sales".

FOR EACH Customer, SalesRep OF Customer BREAK BY Customer.State:
  ACCUMULATE Customer.Balance (TOTAL BY Customer.State).
  IF LAST-OF(Customer.State) THEN DO:
    CREATE showsales.
    showsales.state = Customer.State.
    showsales.tot-sales = ACCUM TOTAL BY Customer.State Customer.Balance.
    showsales.region = SalesRep.Region.
  END.
END.

FOR EACH showsales BREAK BY showsales.region BY showsales.state:
  IF FIRST-OF(showsales.region) THEN
    DISPLAY showsales.region.
  DISPLAY showsales.state tot-sales (TOTAL BY showsales.region).
END.
```

Notes

- You cannot perform a unique find on a work table. When finding records in a work table, you must use FIRST, LAST, NEXT, or PREV with the FIND statement, unless you are finding a record using its ROWID.
- You cannot define a field in a work table with the MEMPTR data type, but you can define a work table field as ROWID or RAW.
- You cannot define shared objects, work tables, or temp-tables within an internal procedure, a method in a class, or a user-defined function.
- ABL disregards the following options when used in conjunction with a work table:
  - The VALIDATE option on a DELETE statement
  - The SHARE-LOCK, EXCLUSIVE-LOCK, and NO-LOCK options used with the FIND or FOR statements

    - The NO-WAIT option on the FIND statement

- When you use the AMBIGUOUS function in conjunction with a work table, the function always returns a value of FALSE.
- Complete work table definitions must be included in a DEFINE SHARED WORK-TABLE statement and shared work tables must be defined identically.
- These are the differences between work tables and regular database tables:
    - The AVM does not use the OpenEdge database manager (and server for multi-user systems) when working with work tables.
    - If you do not explicitly delete the records in a work table, the AVM discards those records, and the work table, at the end of the procedure that initially defined the work table.
    - Users do not have access to each other's work tables.
- Because you cannot index a work table, the AVM uses the following rules for storing records in a work table:
    - If you create a series of work table records without doing any other record operations, the AVM orders the newly created records in the order they were entered.
    - If you use the FIND PREV statement at the beginning of a work table and then create a work table record, the AVM stores that record at the beginning of the work table.
    - When you use the FIND statement to find a work table record and then use the CREATE statement to create a new work table record, the AVM stores that new record after the record you just found.
- Data handling statements that cause the AVM to automatically start a transaction for a regular table will not cause the AVM to automatically start a transaction for a work table. To start a transaction for operations involving a work table, Use the TRANSACTION keyword.
- Work tables are private:
    - Even if two users define work tables with the same name, the work tables are private; one user cannot see records the other user has created.
    - If two procedures run by the same user define work tables with the same name, the AVM treats those work tables as two separate tables unless the SHARED option is included in both procedures.
- DEFINE SHARED WORK-TABLE does not automatically provide a shared buffer. If you want to use a shared buffer with a shared work table, you must define that buffer.
- Work table records are built in 64-byte sections. Approximately the first 60 bytes of each record are taken up by record specification information (or a record header). That is, if a record is 14 bytes long, it will be stored in two 64-byte sections, using the first 60 bytes as a record header. If the record is 80 bytes long, it will fit into three 64-byte sections. The first part contains 60 bytes of header information plus the first 4 bytes of the record. The second section contains 64 bytes of the record. And the last section contains the remaining record bytes.
- The NO-UNDO option in a work table definition overrides a transaction UNDO for CREATE, UPDATE, DELETE, and RELEASE statements accessing the work table, regardless of whether these statements are executed before or during the transaction block that is undone.
- A transaction UNDO overrides a FIND statement accessing a work table defined with the NO-UNDO option, regardless of whether the find is executed before or during the transaction that is undone.

  You should use the CASE-SENSITIVE option only when it is important to distinguish between uppercase and lowercase values entered for a character field. For example, use CASE SENSITIVE to define a field for a part number that contains mixed upper case and lowercase characters.

- A SHARED work table remains in scope for an instance of a persistent procedure until the instance is deleted. This is true even if the original procedure that defined the work table as NEW SHARED goes out of scope while the procedure instance remains persistent.

  If a trigger or internal procedure of a persistent procedure executes an external subprocedure that defines a SHARED work table, ABL includes the persistent procedure in the resolution of the corresponding NEW SHARED work table as though the procedure were on the procedure call stack.

- You cannot define a SHARED or NEW SHARED work table in a class definition (`.cls`) file. If you do, ABL generates a compilation error.
- You can specify a join between a temp-table or work table and any appropriate table using the OF keyword. The two tables must contain a commonly named field that participates in a unique index for at least one of the tables. For more information on table joins see the Record phrase reference entry.
- See *Develop ABL Applications* for information on work tables and temp-tables.

See also

{ } Argument reference, { } Include file reference, Class-based data member access, CREATE statement, DEFINE BUFFER statement, DEFINE TEMP-TABLE statement, FIND statement, Format phrase, RUN statement

DEFINE WORKFILE statement

See the DEFINE WORK-TABLE statement reference entry.

This statement is supported only for backward compatibility. Use the DEFINE TEMP-TABLE statement, instead.

Note: Does not apply to SpeedScript programming.

Syntax

```
DEFINE {[[ NEW ] SHARED ]|[ PRIVATE ]}
  { WORK-TABLE | WORKFILE }work-table-name
  [ NO-UNDO ]
  [ LIKE tablename]
  [ FIELD field-name
      {     AS data-type
          | LIKE field
      }
      [field-options]
  ]...
```

DEFINED preprocessor function

Returns the status of a preprocessor name or include file argument name as an INTEGER value. You can use the DEFINED function only within a preprocessor &IF expression.

Syntax

```
DEFINED ( name )
```

name

Preprocessor name or include file argument name whose status you want to check. You do not specify *name* as a preprocessor name reference or include file argument reference. That is, it is not quoted and does not appear in the reference form, {&*name*}. For example if you had a preprocessor name MAX-EXPENSE, the argument would appear as follows:

```
DEFINED(MAX-EXPENSE)
```

Notes

This function returns a value of 1 if the argument was a name defined with the &GLOBAL-DEFINE directive; a value of 2 if the argument was passed as an include file parameter; and a value of 3 if the argument was a name defined with the &SCOPED-DEFINE directive. If the argument was not defined and was not an include file parameter, then this function returns a value of 0. The value returned refers to the definition that is current at the point of the call.

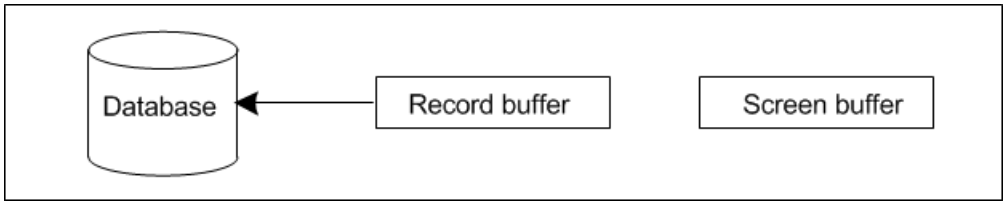
See also

{ } Include file reference, &GLOBAL-DEFINE preprocessor directive, &SCOPED-DEFINE preprocessor directive

DELETE statement

Removes a record from a record buffer and from the database.

Data movement



Syntax

```
DELETE record
   [ VALIDATE ( condition , msg-expression ) ]
   [ NO-ERROR ]
```

record

The name of a record buffer. You can delete a record only after it has been put into a record buffer by a CREATE, FIND, FOR EACH, or INSERT statement.

If you define an alternate buffer for a table, you can delete a record from that buffer by using the name of the buffer with the DELETE statement.

To delete a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

VALIDATE (*condition, msg-expression*)

Use the VALIDATE option to specify a logical value that allows the deletion of a record when TRUE, but does not allow the deletion of a record when FALSE.

The *condition* is a Boolean expression (a constant, field name, variable name, or expression) with a value of TRUE or FALSE.

The *msg-expression* is the message you want to display if the *condition* is FALSE. You must enclose *msg-expression* in quotation marks ("").

You can also describe delete validation criteria for a table in the Data Dictionary. To suppress the Data Dictionary delete validation criteria for a table, use the VALIDATE option as follows:

```
VALIDATE(TRUE,"")
```

If you use the DELETE statement to delete a record in a work table, the AVM disregards any VALIDATE option you use with the DELETE statement.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Examples

The r-delet.p procedure deletes all the records in the Customer table.

r-delet.p

```
FOR EACH Customer:
  DELETE Customer.
END.
```

The r-delet2.p procedure prompts the user for a Customer number and then displays the name of that Customer. It then prompts the user to press y to confirm the deletion of the Customer record. The user's response is stored in the del variable. If the value of the del variable is y, the procedure deletes the Customer record.

r-delet2.p

```
DEFINE VARIABLE del AS LOGICAL NO-UNDO FORMAT "y/n".

REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum.
  DISPLAY Customer.Name.
  del = NO.
  UPDATE del LABEL "Enter ""y"" to confirm delete".
  IF del THEN DELETE Customer.
END.
```

The `r-delval.p` procedure prompts the user for a Customer number. The procedure displays the name of the Customer and prompts the user: Do you want to delete this Customer? If the user answers no, the procedure prompts the user for another Customer number. If the user answers yes, the procedure checks whether the Customer has orders, using the `VALIDATE` option. If they do have orders, the procedure displays this message: This Customer has outstanding orders and cannot be deleted. If the Customer has no orders, the procedure deletes the Customer.

r-delval.p

```
DEFINE VARIABLE ans AS LOGICAL NO-UNDO.

REPEAT WITH 1 DOWN:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum.
  DISPLAY Customer.Name.
  ans = NO.
  DISPLAY "Do you want to delete this Customer ?"
    WITH FRAME f-query.
  UPDATE ans WITH FRAME f-query NO-LABELS.
  IF ans THEN
    DELETE Customer VALIDATE(NOT(CAN-FIND(Order OF Customer)),
    "This Customer has outstanding orders and cannot be deleted.").
END.
```

Notes

- When you run procedures that delete large numbers of records (for example, a month-end table purge), the process runs much faster if you use the No Crash Protection (-i) parameter in single-user mode. (You must back up your database before using this option.) See *Startup Command and Parameter Reference* for more information on startup parameters.
- Deleting records does not change the amount of space the database takes up on the disk. The AVM re-uses ROWIDs. It does not delete the ROWID when a record is deleted. To recover disk space, you must dump and reload your database.
- The `DELETE` statement causes any related database DELETE triggers to execute. All DELETE triggers execute before the AVM actually deletes the record. While a DELETE trigger is executing, all `FIND` requests for the record (even within the trigger) fail, as if the record were already deleted. If a DELETE trigger fails (or executes a RETURN ERROR statement), the corresponding record is not deleted.
- If a table has both a DELETE trigger and delete VALIDATION, the DELETE trigger executes before the validation is performed.
- If you have previously retrieved *`record`* with a field list, the `DELETE` statement rereads the complete record before deleting it.

See also

CREATE statement, FIND statement, FOR statement, INSERT statement, NO-ERROR option

DELETE ALIAS statement

Deletes an alias from the alias table.

Syntax

```
DELETE ALIAS {alias | VALUE ( expression ) }
```

alias

An existing alias. It can be an unquoted string or a quoted string.

VALUE (*expression*)

A character-string expression that evaluates to an existing alias.

Example

This procedure deletes the alias myalias from the alias table:

r-dalias.p

```
DELETE ALIAS myalias.
```

Notes

- If a precompiled program requires an alias and you delete that alias, the program will not run.
- If you try to delete a nonexistent alias, nothing happens.

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

DELETE OBJECT statement

Deletes an object such as a widget, a procedure, a server, a socket, or an instance of a class. Deleting the object causes all allocated resources associated with the object to be returned to the system (except when otherwise noted).

Syntax

```
DELETE OBJECT { handle | object-reference } [ NO-ERROR ]
```

handle

A handle to an ABL handle-based object. The handle argument must be a variable of type HANDLE and must contain a valid handle.

If the handle parameter refers to a widget, the DELETE OBJECT statement is a synonym for the DELETE WIDGET statement.

If the handle parameter refers to a persistent procedure handle or proxy procedure handle, the DELETE OBJECT statement is a synonym for the DELETE PROCEDURE statement. This statement deletes a local persistent procedure handle immediately. For a proxy procedure handle, this statement deletes the handle immediately unless there is an outstanding asynchronous request on this handle (*handle*:ASYNC-REQUEST-COUNT is greater than zero (0)). If *handle*:ASYNC-REQUEST-COUNT is greater than zero (0), this statement raises the ERROR condition. Otherwise, the statement also sends a request to the application server to delete the corresponding remote persistent procedure on the application server. If the application server is executing any asynchronous requests ahead of it, the AVM queues the delete request (as with any asynchronous remote request) until the application server is available to handle it.

Note: This same behavior occurs if the remote procedure deletes itself (using DELETE...THIS-PROCEDURE) on the application server.

If the handle parameter refers to a server handle, the DELETE OBJECT statement:

- Checks that the handle parameter refers to a valid server handle, and that the handle parameter's CONNECTED attribute is FALSE (no application server is connected to it). If one of these checks fails, the statement raises the ERROR condition.
- Deletes the handle immediately, if the server handle is valid, unless there is an outstanding asynchronous request on this handle (*handle*:ASYNC-REQUEST-COUNT is greater than zero (0)). If there is an outstanding asynchronous request, this statement raises the ERROR condition.

Deleting a server handle removes the handle from the server handle chain of the SESSION system handle, and resets SESSION:FIRST-SERVER and SESSION:LAST-SERVER if necessary. This also deletes all of the asynchronous request handles associated with the server and then deletes the server object.

If *handle* refers to an asynchronous request handle, the DELETE OBJECT statement takes one of the following actions:

- If the *handle*:COMPLETE attribute is FALSE, it raises the ERROR condition.
- If the *handle*:COMPLETE attribute is TRUE, it removes *handle* from the chain of asynchronous request handles referenced by the FIRST-ASYNC-REQUEST and the LAST-ASYNC-REQUEST attributes of the server handle, and deletes *handle*.

If this is a socket handle, the application must disconnect the socket from a port using the DISCONNECT( ) method before a socket object can be deleted. The DELETE OBJECT statement raises ERROR if an application deletes a socket object that is still associated with a port.

If this is a server socket handle, the application must call DISABLE-CONNECTIONS( ) before a server socket object can be deleted. The DELETE OBJECT statement raises ERROR if an application deletes a server socket object that is still listening for connections.

Where *handle* refers to a dynamic buffer, it is recommended practice to execute the DELETE OBJECT statement at the same level (that is, as part of the same sub-transaction) as the transaction that created the buffer.

object-reference

An object reference to a class instance. The object reference argument must be an ABL object reference variable, such as one defined using the VAR statement, DEFINE VARIABLE statement, or the DEFINE PARAMETER statement with the CLASS option, and it must contain a valid object reference.

Note: You can validate an object reference by using the VALID-OBJECT function.

When you delete a class instance, the AVM invokes the destructor for the class and the destructor for each class in its inherited class hierarchy, if any. The destructor can be used for freeing resources allocated to the object when the object is deleted. At this time, the object context goes out of scope. In addition, the object is removed from the list of valid ABL class instances (session object chain) referenced by the FIRST-OBJECT attribute or LAST-OBJECT attribute of the SESSION system handle.

If you do not delete a class instance and you have not turned off automatic garbage collection using the No Garbage Collection (-nogc) startup parameter, the instance is deleted when there are no more references to it.

OpenEdge includes a performance tuning feature for ABL class-based applications that controls how the AVM deletes objects. The Re-usable Objects Cache (-reusableObjects) startup parameter specifies the number of deleted class objects that the AVM stores for later re-initialization. By default, -reusableObjects is set to 25. When you use -reusableObjects, the AVM transfers the deleted object for most ABL classes to a re-usable objects cache. If your application causes the AVM to instantiate the same class later, the stored object is re-initialized and removed from the cache. The re-initialized object has a new UUID and the same initial data as a new instance of the class. The re-use of the object saves much of the overhead of instantiating a class.

Note: The cache does not store .NET classes, .NET-derived ABL classes, classes with static elements, or classes compiled during your session.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Notes

- When a temp-table object is passed as a TABLE-HANDLE output parameter, the deletion of the object following the DELETE OBJECT statement is postponed until the procedure goes out of scope. When the procedure returns, the TABLE-HANDLE is created, receives a copy of the original temp-table, and is then returned.
- The OUTPUT TABLE-HANDLE parameter creates a TEMP-TABLE in the target procedure, which is added to the SESSION widget-pool. You must use the DELETE OBJECT statement to delete this TEMP-TABLE when it is no longer needed, or you will create a memory leak.
- Using automatic garbage collection (the default), the AVM deletes each class instance when no references to the object exist other than on the session object chain (accessible using the FIRST-OBJECT attribute or LAST-OBJECT attribute of the SESSION system handle). When a class instance is garbage collected, its destructor runs as if you deleted the object using DELETE OBJECT. As with DELETE OBJECT, automatic garbage collection stores the deleted object in a re-usable object cache.

- You can turn off automatic garbage collection using the No Garbage Collection (`-nogc`) startup parameter. However, if you do so, you must decide when to use this statement to delete each class instance you no longer need. Furthermore, a class instance that has no other references still remains in memory. You must locate it by walking the session object chain and then delete it.
- The Re-usable Object Cache (`-reusableObjects`) startup parameter controls how many deleted class-based objects are cached. By default, the parameter is set to 25. You can disable this feature by setting the parameter value to zero. When the cache is full and you delete another object, the AVM uses a least-recently used scheme to make room in the cache.
- The AVM only uses an object in the re-usable object cache to re-instantiate the same class. It cannot use an object to instantiate another class in the same inheritance hierarchy. For example, Class B inherits Class A and is inherited by Class C. If the cache contains an instance of Class B, the AVM cannot use that object to help instantiate either an instance of Class A or Class C. The AVM can only use that object for a new instance of Class B.
- If you invoke `DELETE OBJECT` from within a class constructor, specifying the THIS-OBJECT system reference for *`object-reference`*, class instantiation stops, and the AVM invokes the destructor for the instantiating class and for each class in its class hierarchy where the constructor has already completed execution. After all destructors have completed, control returns to the statement following the statement that instantiated the class using the NEW function (classes), the NEW statement, or the DYNAMIC-NEW statement. When control returns from the context of a `NEW` statement or a `DYNAMIC-NEW` statement, the data element set to receive the object reference to the failed class instantiation receives the Unknown value (?).
- When you invoke `DELETE OBJECT` from within a class constructor, the object is stored to the re-usable object cache. By the time the instantiation reaches a constructor, the object data stored in the cache is already in memory. Even though the current instantiation never completes, the AVM can use the object data to instantiate another instance of the class at a later time without problems.
- If you need to terminate class instantiation within a constructor, Progress Software Corporation recommends that you use a `RETURN ERROR` or a `THROW` (as in the RETURN statement or UNDO statement, respectively). This has the same effect as using the `DELETE OBJECT THIS-OBJECT` statement and also raises `ERROR` on the statement that attempts to instantiate the class. This allows your application to handle the uninstantiated class condition using ABL error handling options, such as the CATCH statement or the `NO-ERROR` option on many statements.
- For more information on working with socket and server socket objects, see *OpenEdge Programming Interfaces*.
- For more information on working with class instances, see *Develop Object-oriented ABL Applications*.
- Automatic garbage collection works for .NET objects, or for ABL objects that derive from .NET objects, similar to how it works for pure ABL class instances. However, .NET objects and the .NET components of an ABL-derived .NET object are also managed separately by .NET garbage collection. Thus, the release of memory held by a .NET object or an ABL-derived .NET object depends on both ABL and .NET garbage collection.
- For a .NET form object, you can typically invoke its `Close( )` method or the `System.Windows.Forms.Application:Exit( )` method (for non-modal forms) to both close the form and delete the associated object, making it available for .NET garbage collection. One instance where a form is not deleted using the `Close( )` method is when a non-modal form is an invisible child form of a multiple-document interface (MDI). In this case, you must also invoke the `DELETE OBJECT` statement on the associated object reference, or otherwise remove all references to the form, to make the form available for .NET garbage collection.
- In a .NET event handler for the `FormClosing` event, if you do delete object references, do not delete the object reference to the form (sender) that published the `FormClosing` event. Otherwise, this causes the `FormClosing` event to be published a second time for the same form. If you need to delete the sender of an event associated with the closing of a .NET form, execute the `DELETE OBJECT` statement on the sender

in an event handler for the FormClosed event of the form. However, note that with garbage collection, you typically do not need to delete any object references in a .NET event handler.

- In ABL, the .NET Dispose( ) method is also available for freeing resources held by a .NET object. Its main purpose is to clean up unmanaged resources, such as file handles or socket connections. A well-behaved object cleans these up in its destructor, but calling Dispose( ) forces this cleanup in a more timely manner. It is good practice to call Dispose( ) on any object that implements the IDisposable interface.

Note: .NET automatically calls Dispose( ) on a non-modal form when it is closed, unless the form is shown as a dialog box. .NET simply hides the dialog when it is closed. However, the AVM calls Dispose( ) by default for dialogs that inherit from Progress.Windows.Form. The AVM does not do this if the DisposeDialogOnClose property of the form is set to FALSE. This can be set on a particular instance or globally using the No Dialog Dispose (-nodialogdispose) startup parameter. One use case for not calling Dispose( ) is if you want to re-use the form at a later time.

See also

DELETE PROCEDURE statement, DELETE WIDGET statement, DESTRUCTOR statement, ERROR-STATUS system handle, THIS-OBJECT system reference, NO-ERROR option, RETURN statement, UNDO statement

DELETE PROCEDURE statement

Deletes an instance of a persistent procedure. The persistent procedure can be local or remote.

Syntax

```
DELETE PROCEDURE proc-handle [ NO-ERROR ]
```

proc-handle

The handle of a local or remote persistent procedure. This is a variable, field, or expression of type HANDLE that contains a valid persistent procedure handle.

For a proxy procedure handle, this statement deletes the handle immediately unless there is an outstanding asynchronous request on this handle (*handle*:ASYNC-REQUEST-COUNT is greater than zero (0)). If *handle*:ASYNC-REQUEST-COUNT is greater than zero (0), this statement raises the ERROR condition. Otherwise, the statement also sends a request to the application server to delete the corresponding remote persistent procedure on the application server. If the application server is executing any asynchronous requests ahead of it, the AVM queues the delete request (as with any asynchronous remote request) until the application server is available to handle it.

Note: This same behavior occurs if the remote procedure deletes itself (using DELETE...THIS-PROCEDURE) on the application server.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

When you run the following procedure non-persistently, the procedure creates a persistent instance of itself in addition to the non-persistent instance, creating two query windows for the Customer table. Choosing the Cancel button in either window causes the instance that owns that window to terminate. If the instance you terminate is persistent, the Cancel button runs an internal procedure that executes the DELETE PROCEDURE statement for that instance as specified by the THIS-PROCEDURE system handle.

r-delprc.p

```
DEFINE QUERY custq FOR Customer.
DEFINE BROWSE custb QUERY custq
  DISPLAY name balance phone WITH 10 DOWN.
DEFINE BUTTON bName LABEL "Query on Name".
DEFINE BUTTON bBalance LABEL "Query on Balance".
DEFINE BUTTON bCancel LABEL "Cancel".
DEFINE FRAME CustFrame custb SKIP bName bBalance bCancel.
DEFINE VARIABLE custwin AS HANDLE.

ON CHOOSE OF bName IN FRAME CustFrame DO:
  custwin:TITLE = "Customers by Name".
  OPEN QUERY custq FOR EACH Customer BY Customer.Name.
END.

ON CHOOSE OF bBalance IN FRAME CustFrame DO:
  custwin:TITLE = "Customers by Balance".
  OPEN QUERY custq FOR EACH Customer BY Customer.Balance DESCENDING.
END.

IF THIS-PROCEDURE:PERSISTENT THEN DO:
  THIS-PROCEDURE:PRIVATE-DATA = "Customer Browse".
  CREATE WIDGET-POOL.
END.

CREATE WINDOW custwin ASSIGN
  TITLE            = IF THIS-PROCEDURE:PERSISTENT THEN
    "Persistent Customer Browser" ELSE "Customer Browser"
  SCROLL-BARS      = FALSE
  MAX-HEIGHT-CHARS = FRAME CustFrame:HEIGHT-CHARS
  MAX-WIDTH-CHARS  = FRAME CustFrame:WIDTH-CHARS.

THIS-PROCEDURE:CURRENT-WINDOW = custwin.
ENABLE ALL WITH FRAME CustFrame.
IF THIS-PROCEDURE:PERSISTENT THEN DO:
  ON CHOOSE OF bCancel IN FRAME CustFrame DO:
    RUN destroy-query.
  END.
END.
ELSE DO:
  RUN r-delprc.p PERSISTENT.
  WAIT-FOR CHOOSE OF bCancel IN FRAME CustFrame.
  DELETE WIDGET custwin.
END.

PROCEDURE destroy-query:
  DELETE PROCEDURE THIS-PROCEDURE.
  DELETE WIDGET-POOL.
END PROCEDURE.
```

Notes

- To be valid for deletion, *proc-handle* must reference an active persistent procedure. You can use the VALID-HANDLE function and PERSISTENT procedure attribute to check the validity of *proc-handle*. Thus, both VALID-HANDLE(*proc-handle*) and *proc-handle*:PERSISTENT must be TRUE to delete

the specified procedure. If either of these expressions is FALSE, the DELETE PROCEDURE statement raises the ERROR condition.

- When you delete a persistent procedure instance, its context goes out of scope and all allocated resources are returned to the system. In addition, it is removed from the chain of persistent procedures referenced by the FIRST-PROCEDURE and LAST-PROCEDURE attributes of the SESSION system handle.
- If you delete a persistent procedure instance while executing statements within that procedure, the DELETE PROCEDURE statement pends until the largest executing block in the persistent procedure terminates. Thus, if the DELETE PROCEDURE occurs while the main procedure block is executing (when the persistent procedure is called), the procedure is deleted when the procedure returns (as if it were non-persistent). If the DELETE PROCEDURE occurs during execution of a trigger or execution of an internal procedure that is called from another external procedure, the procedure is deleted after the trigger block or internal procedure returns. Note that while the delete is pending, the persistent procedure remains valid in the persistent procedure chain.
- The DELETE PROCEDURE statement will only delete the proxy handle on the client of a procedure run as single-run or singleton. This is because the AVM automatically deletes a single-run procedure after execution, and a singleton procedure may be running on multiple Progress Application Server (PAS) for OpenEdge instances.
- The DELETE PROCEDURE statement disconnects any local buffers established by the procedure. In addition, any buffers passed as parameters to a persistent procedure are treated as local buffers. While all cursor positioning established on these buffers by the persistent procedure is lost, there is no affect on the original buffers passed as parameters from the caller. Note that all buffers are validated before being disconnected (which might cause database write triggers to execute). If the validation fails, the DELETE PROCEDURE statement raises the ERROR condition and pends the deletion until the validation succeeds and all database write triggers have completed.

See also

RUN statement, THIS-PROCEDURE system handle, NO-ERROR option, VALID-HANDLE function

DELETE WIDGET statement

Deletes one or more dynamic widgets.

Syntax

```
DELETE WIDGET handle [handle] . . .
```

handle

The handle of a dynamic widget.

Example

In the following example, the DELETE WIDGET statements deletes the dynamic button that you select:

r-delwid.p

```
DEFINE VARIABLE wh-tmp AS HANDLE  NO-UNDO.
DEFINE VARIABLE ix     AS INTEGER NO-UNDO.

DEFINE BUTTON b_quit LABEL "Quit" AUTO-ENDKEY.
DEFINE BUTTON b_make LABEL "Make Buttons".

DEFINE FRAME butt-frame b_make b_quit
  WITH CENTERED ROW 2.
DEFINE FRAME new-buttons
  WITH WIDTH 48 CENTERED TITLE "New Buttons".

FRAME new-buttons:HEIGHT-CHARS = 3.

ON CHOOSE OF b_make IN FRAME butt-frame DO:
  DISABLE b_make WITH FRAME butt-frame.
  DO ix = 1 TO 10:
    CREATE BUTTON wh-tmp ASSIGN
      FRAME     = FRAME new-buttons:HANDLE
      COLUMN    = ix * 4
      LABEL     = STRING(ix)
      SENSITIVE = TRUE
      VISIBLE   = TRUE
      TRIGGERS:
        ON CHOOSE
          PERSISTENT RUN del-self.
      END.
  END.
END.

ENABLE b_make b_quit WITH FRAME butt-frame.

WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame.

PROCEDURE del-self:
  IF SELF:PREV-SIBLING = ? AND SELF:NEXT-SIBLING = ? THEN DO:
    HIDE FRAME new-buttons.
    ENABLE b_make WITH FRAME butt-frame.
  END.
  MESSAGE "Deleting Widget, Button" SELF:LABEL.
  DELETE WIDGET SELF.
END PROCEDURE.
```

Notes

- If you do not explicitly delete a dynamically created widget, it is deleted when its widget pool is deleted. If you do not create a new unnamed widget pool and do not explicitly specify a named widget pool when you create the widgets, all dynamic widgets are placed in the session pool. The session pool is not deleted until the ABL session that created it ends.
- If *handle* refers to a control-frame, any ActiveX control associated with the widget is also deleted. For more information on ActiveX support in ABL, see *OpenEdge Programming Interfaces*.
- For SpeedScript, use with buffer-field, buffer-object, buffer, and query-object handles.

See also

CREATE widget statement

DELETE WIDGET-POOL statement

Deletes a defined widget pool.

Note: Does not apply to SpeedScript programming.

Syntax

```
DELETE WIDGET-POOL [ pool-name ] [ NO-ERROR ]
```

pool-name

The name of a defined dynamic widget pool. If you omit *pool-name*, the statement deletes the unnamed pool most recently created in the current or a calling procedure or class-based method.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

The following example creates a named widget pool and lets you add buttons to it. When you choose Delete Buttons, the widget pool is deleted. (Therefore all the buttons in the pool are also deleted.) Similarly, when you choose Quit to exit the procedure the widget pool is also deleted. Because the pool is persistent, it remains allocated for the rest of your session if you do not delete it.

r-widpl.p

```
DEFINE VARIABLE wh AS HANDLE NO-UNDO.

DEFINE BUTTON b_create LABEL "Create Button".
DEFINE BUTTON b_del    LABEL "Delete Buttons".
DEFINE BUTTON b_quit   LABEL "Quit"
  TRIGGERS:
    ON CHOOSE DO:
      IF VALID-HANDLE(wh) THEN
        DELETE WIDGET-POOL "new-buttons".
      QUIT.
    END.
  END.

DEFINE FRAME butt-frame
  b_create b_del b_quit
  WITH ROW SCREEN-LINES - 2.

DEFINE FRAME new-buttons
  WITH SIZE 76 BY 11 CENTERED ROW 2 TITLE "New Buttons".

ON CHOOSE OF b_create IN FRAME butt-frame DO:
  STATUS INPUT "Press RETURN to select a new button".
  IF wh = ? OR NOT VALID-HANDLE(wh) THEN
    CREATE WIDGET-POOL "new-buttons" PERSISTENT.
  CREATE BUTTON wh IN WIDGET-POOL "new-buttons" ASSIGN
    FRAME     = FRAME new-buttons:HANDLE
    ROW       = RANDOM(2, 9)
    COLUMN    = RANDOM(2, 58)
    LABEL     = "BUTTON " + STRING(etime)
    SENSITIVE = TRUE
    VISIBLE   = TRUE
    MOVABLE   = TRUE
    TRIGGERS:
      ON CHOOSE PERSISTENT RUN dispmsg.
    END.
END.

ON CHOOSE OF b_del IN FRAME butt-frame DO:
  IF VALID-HANDLE(wh) THEN
    DELETE WIDGET-POOL "new-buttons".
  STATUS INPUT.
END.

ENABLE b_create b_del b_quit WITH FRAME butt-frame.
WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame.

PROCEDURE dispmsg:
  MESSAGE "You chose button " SELF:LABEL.
END PROCEDURE.
```

Notes

- When you delete a widget pool, all widgets in that pool are automatically deleted.
- If you do not delete a non-persistent widget pool, it is deleted when the procedure or method that created it ends. If you do not delete a persistent widget pool, it is deleted when the session ends.
- All named widget pools are globally scoped. While a named widget pool is allocated, any procedure or method within the same process can access that widget pool. If you try to delete a named widget pool that does not exist, the AVM raises the ERROR condition.

See also

CREATE WIDGET-POOL statement, DELETE WIDGET statement, NO-ERROR option

DESTRUCTOR statement

Defines a destructor for a class. A destructor is a special type of method that the AVM invokes when an instance of the class is deleted, either manually using the DELETE OBJECT statement or automatically through garbage collection.

Note: This statement is applicable only when used in a class definition (`.cls`) file.

Syntax

```
DESTRUCTOR [ PUBLIC ] class-name ( ) :
destructor-body
```

[PUBLIC]

Specifies the access mode for the destructor. For destructors, this is always PUBLIC by default.

As a PUBLIC resource, a destructor can be accessed indirectly by the defining class, any of its inheriting classes, and any class or procedure that instantiates the class object (that is, through an object reference), either by deleting the object instance using the DELETE OBJECT statement or as a result of garbage collection.

class-name

The name of the class this method destroys. This name must match the class name portion of the type name for the class (that is, the name of the class definition file excluding the `.cls` extension and any package path information).

destructor-body

The body of the destructor definition. Define the destructor body using the following syntax:

```
      .
      .
      .
  method-logic      .
      .
      .
      .
END [ DESTRUCTOR ].
```

method-logic

The logic of the destructor, which can contain any ABL statements currently allowed within a PROCEDURE block including class-related statements, but excluding the RETURN ERROR statement.

This method typically contains logic to release system resources used by the class instance, for example, by executing the DELETE OBJECT statement for handle-based objects created by the class or by invoking the Dispose( ) method on .NET objects created by the class. However, note that if the ABL class inherits from a .NET class, you cannot reliably access members of the .NET super class from your class destructor, because .NET might already have garbage collected it.

END [DESTRUCTOR]

Specifies the end of the destructor body definition. You must end the destructor body definition with the END statement.

Example

The following example shows the definition of a destructor:

```
DESTRUCTOR PUBLIC CustObj( ):

  EMPTY TEMP-TABLE ttCust.

END DESTRUCTOR.
```

Notes

- You can terminate a DESTRUCTOR statement with either a period (.) or a colon (:), but typically use a colon (:).
- A destructor has no parameters and no return value.
- You never explicitly invoke the destructor to delete a class instance. The method is implicitly invoked when the object is destroyed manually by the DELETE OBJECT statement or automatically as a result of garbage collection.
- The AVM also invokes the destructor of a given class if some constructor in the same class hierarchy raises ERROR during object instantiation and the constructor for the given class has already completed execution during this instantiation.
- If your application does not require it, you do not need to define a destructor for the class. ABL provides a default destructor for classes that do not define one.

See also

CLASS statement, CONSTRUCTOR statement, DELETE OBJECT statement

DICTIONARY statement

Runs the OpenEdge Data Dictionary.

Note: Does not apply to SpeedScript programming.

Syntax

```
DICTIONARY
```

Example

This procedure runs the Data Dictionary if the user answers yes to a prompt:

r-dict.p

```
DEFINE VARIABLE ans AS LOGICAL NO-UNDO.

DISPLAY "Do you want to access the Dictionary?"
  WITH ROW 7 COLUMN 20 NO-LABELS.
UPDATE ans.
IF ans THEN DICTIONARY.
```

Notes

- The DICTIONARY statement is equivalent to RUN `dict.p`: it runs the ABL procedure called `dict.p`. The AVM uses the regular search rules to find the dictionary procedure. The dictionary procedure is part of the ABL system software.
- The AVM Query/Run-time provides a restricted version of the Data Dictionary.
- For more information on the Data Dictionary, see its on-line help.

DISABLE statement

Disables input for one or more field-level and child frame widgets within a frame that were previously enabled with the ENABLE statement. Disabling a widget prevents the user from providing input to the widget, but does not remove it from the display.

Syntax

```
DISABLE [ UNLESS-HIDDEN ]
 {     ALL [ EXCEPT field ... ]
   |  { field [ WHEN expression ] }...
 }
 { frame-phrase ] }
```

UNLESS-HIDDEN

Restricts DISABLE to fields whose HIDDEN attribute is FALSE.

`ALL [ EXCEPT field... ]`

Specifies that all field-level widgets for a frame should be disabled, except those that you optionally specify.

`field` `[ WHEN expression ]`

A field-level widget to be disabled. If you use the WHEN option, then the field is disabled only if *`expression`* is TRUE when the DISABLE statement is executed. The *`expression`* must evaluate to a LOGICAL value.

`frame-phrase`

The frame that contains the widgets to disable. If you omit *`frame-phrase`*, the default frame for the current block is assumed.

You cannot use the IN WINDOW option of the frame phrase within a DISABLE statement. For more information on *`frame-phrase`*, see the Frame phrase reference entry.

Example

In the following example, the `CustNum` field and the Quit button are initially active. When you press **GO** in the `CustNum` field, that field becomes disabled and the Save and Undo buttons and the `CreditLimit` field are enabled. If you choose either the Save or Undo button, those buttons and the `CreditLimit` field are again disabled and the `CustNum` field is enabled again.

r-enable.p

```
DEFINE VARIABLE ok AS LOGICAL NO-UNDO.

DEFINE BUTTON b_quit LABEL "Quit" AUTO-ENDKEY.
DEFINE BUTTON b_save LABEL "Save".
DEFINE BUTTON b_undo LABEL "Undo".

DEFINE FRAME butt-frame
  b_save b_undo b_quit
  WITH CENTERED ROW SCREEN-LINES - 1.

FORM Customer
  WITH FRAME cust-info SIDE-LABELS CENTERED
  TITLE "Update Customer Credit Limit".

ON CHOOSE OF b_save, b_undo IN FRAME butt-frame DO:
  DISABLE b_save b_undo WITH FRAME butt-frame.
  DISABLE Customer.CreditLimit WITH FRAME cust-info.
  ENABLE Customer.CustNum WITH FRAME cust-info.
  IF SELF:LABEL = "save" THEN
    ASSIGN FRAME cust-info Customer.CreditLimit.
  CLEAR FRAME cust-info NO-PAUSE.
  APPLY "ENTRY" TO Customer.CustNum IN FRAME cust-info.
END.

ON GO OF Customer.CustNum IN FRAME cust-info DO:
  FIND Customer USING Customer.CustNum EXCLUSIVE NO-ERROR.
  IF AVAILABLE Customer THEN DO:
    DISABLE Customer.CustNum WITH FRAME cust-info.
    ENABLE Customer.CreditLimit WITH FRAME cust-info.
    ENABLE ALL WITH FRAME butt-frame.
    DISPLAY Customer WITH FRAME cust-info.
  END.
  ELSE DO:
    MESSAGE "No Customer Record exist for Customer number"
      INPUT Customer.CustNum ", Please re-enter."
      VIEW-AS ALERT-BOX WARNING BUTTONS OK-CANCEL UPDATE OK.
    IF NOT ok THEN
      APPLY "CHOOSE" TO b_quit IN FRAME butt-frame.
  END.
END.

ENABLE Customer.CustNum WITH FRAME cust-info.
ENABLE b_quit WITH FRAME butt-frame.
WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame
   FOCUS Customer.CustNum IN FRAME cust-info.
```

Notes

If you invoke the DISABLE statement for the parent frame of a frame family, the field representation widgets and descendant frames owned by the parent frame are all disabled. However, the field representation widgets of the descendant frames remain enabled and appear sensitive, although they cannot accept input. To disable field representation widgets in the descendant frames and make them appear insensitive, you must invoke DISABLE statements for each of the descendant frames.

See also

ENABLE statement, WAIT-FOR statement (ABL only)

DISABLE TRIGGERS statement

Disables database triggers before you perform a dump or load procedure. You must have CAN-DUMP and CAN-LOAD permissions on the table for which you want to disable the triggers.

Syntax

```
DISABLE TRIGGERS FOR { DUMP | LOAD } OF table-name
  [ ALLOW-REPLICATION ]
```

DUMP

Disabling triggers for DUMP disables the trigger associated with the FIND event for the named table.

LOAD

Disabling triggers for LOAD disables all triggers associated with the CREATE, WRITE, DELETE, REPLICATION-CREATE, REPLICATION-WRITE, REPLICATION-DELETE, and ASSIGN events for the named table.

table-name

The name of the table for which you want to disable the triggers. You can name only one table.

ALLOW-REPLICATION

Tells DISABLE TRIGGERS to disable only CREATE, ASSIGN, and WRITE triggers, and not REPLICATION-CREATE, REPLICATION-DELETE, and REPLICATION-WRITE triggers. The DELETE trigger is not disabled.

For more information on database replication, see the reference entry for the RAW-TRANSFER statement, and *Manage the OpenEdge Database*.

Example

The following example lets you dump or load the contents of a database table. The procedure uses the DISABLE TRIGGERS statement to disable the appropriate triggers before each dump or load operation.

r-dstrig.p

```
DEFINE SUB-MENU file
  MENU-ITEM viewit LABEL "&View Data"
  MENU-ITEM dumpit LABEL "&Dump Data"
  MENU-ITEM loadit LABEL "&Load Data".
  MENU-ITEM exit   LABEL "E&xit".

DEFINE MENU mbar MENUBAR
  SUB-MENU file LABEL "&File".

DEFINE BUTTON b_more LABEL "Next".
DEFINE BUTTON b_exit LABEL "Cancel".

DEFINE VARIABLE ix AS INTEGER NO-UNDO.
```

```
DEFINE FRAME cust-frame
  Customer.CustNum SKIP
  Customer.Name  SKIP
  Customer.Phone SKIP
  b_more b_exit
  WITH CENTERED SIDE-LABELS ROW 3.

DEFINE STREAM cust.

PAUSE 0 BEFORE-HIDE.

ON CHOOSE OF b_exit IN FRAME cust-frame DO:
  HIDE FRAME cust-frame NO-PAUSE.
  DISABLE ALL WITH FRAME cust-frame.
  LEAVE.
END.

ON CHOOSE OF b_more IN FRAME cust-frame DO:
  FIND NEXT Customer NO-LOCK NO-ERROR.
  IF NOT AVAILABLE(Customer) THEN
    RETURN.
  DISPLAY Customer.CustNum Customer.Name Customer.Phone
    WITH FRAME cust-frame.
END.

ON CHOOSE OF MENU-ITEM viewit DO:
  ENABLE ALL WITH FRAME cust-frame.
  FIND FIRST Customer NO-LOCK NO-ERROR.
  DISPLAY Customer.CustNum Customer.Name Customer.Phone
    WITH FRAME cust-frame.
END.

ON CHOOSE OF MENU-ITEM dumpit DO:
  DISABLE TRIGGERS FOR DUMP OF Customer.
  ix = 1.
  SESSION:IMMEDIATE-DISPLAY = TRUE.
  OUTPUT STREAM cust TO "Customer.d".
  FOR EACH Customer NO-LOCK:
    EXPORT STREAM cust Customer.
    DISPLAY ix LABEL "Records Processed"
      WITH FRAME rec-info SIDE-LABELS ROW SCREEN-LINES / 2 CENTERED.
    ix = ix + 1.
    PROCESS EVENTS.
  END.
  SESSION:IMMEDIATE-DISPLAY = FALSE.
  OUTPUT STREAM cust CLOSE. /*
  APPLY "ENTRY" TO b_quit IN FRAME butt-frame. */
END.

IF NOT RETRY THEN
  ASSIGN
    CURRENT-WINDOW:MENUBAR = MENU mbar:HANDLE
    CURRENT-WINDOW:VISIBLE = TRUE.
WAIT-FOR CHOOSE OF MENU-ITEM exit.

ON CHOOSE OF MENU-ITEM loadit DO:
  DISABLE TRIGGERS FOR LOAD OF Customer.
  INPUT FROM "Customer.d".
  SESSION:IMMEDIATE-DISPLAY = TRUE.
  REPEAT:
    CREATE Customer.
    IMPORT Customer.
    DISPLAY ix LABEL "Records Processed"
      WITH FRAME rec-info SIDE-LABELS ROW SCREEN-LINES / 2 CENTERED.
    ix = ix + 1.
    PROCESS EVENTS.
  END.
  INPUT CLOSE.
```

```
  SESSION:IMMEDIATE-DISPLAY = FALSE.
END.
```

Notes

- Triggers disabled with the DISABLE TRIGGERS statement remain disabled for the duration of the procedure in which you issued the statement and any subprocedures.
- The OpenEdge Data Dictionary automatically disables the appropriate triggers during data dump and load operations.
- The Disable Delete Trigger (-disabledeltrig) startup parameter disables the DELETE trigger when ALLOW-REPLICATION is on for the DISABLE TRIGGERS statement or the DISABLE-LOAD-TRIGGER() buffer method. For more information about this parameter, see *Startup Command and Parameter Reference*.

See also

ON statement, TRIGGER PROCEDURE statement

DISCONNECT statement

Disconnects the specified database.

Syntax

```
DISCONNECT
  { logical-name | VALUE ( expression ) }
   [ NO-ERROR ]
```

logical-name

A logical database name. It can be an unquoted string or a quoted string. The *logical-name* is previously set, at startup or with a CONNECT statement, by using the Logical Database Name (-ld) parameter. If a logical name was not specified using the -ld parameter, then the physical database filename, without the .db suffix, is the default logical name.

VALUE (*expression*)

A character-string expression that evaluates to a logical database name.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

This procedure disconnects the database with logical name mydb:

r-discnt.p

```
DISCONNECT mydb.
```

Notes

- By default, the AVM disconnects all databases at the end of a session. The `DISCONNECT` statement, which explicitly disconnects a database, does not execute until all active procedures that reference the database end or stop.
- If a transaction is active for *`logical-name`*, `DISCONNECT` is deferred until the transaction completes or is undone. If a `CONNECT` statement for the same *`logical-name`* database is executed before the same transaction completes or is undone, then the pending `CONNECT` and `DISCONNECT` cancel each other and the database remains connected.
- When the database referred to by *`logical-name`* is disconnected, existing aliases for *`logical-name`* remain in existence. Later, if you connect to a database with the same *`logical-name`*, the same alias is still available.

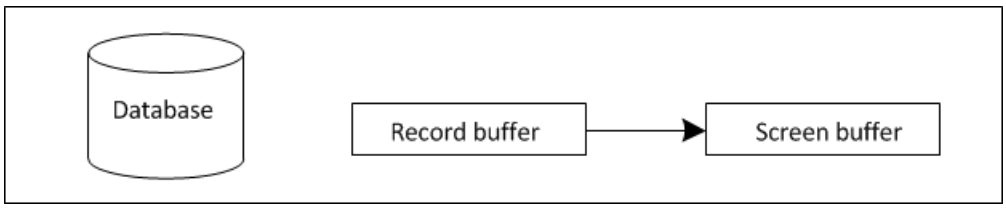
See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, FRAME-DB function, NO-ERROR option, LDBNAME function, PDBNAME function, SDBNAME function

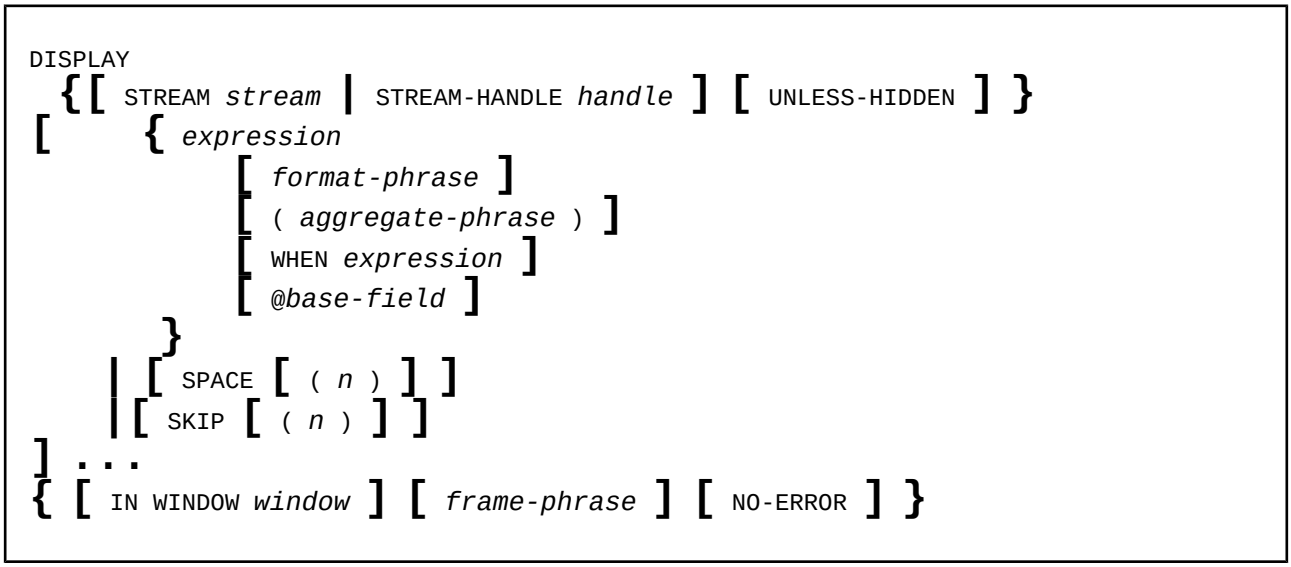
DISPLAY statement

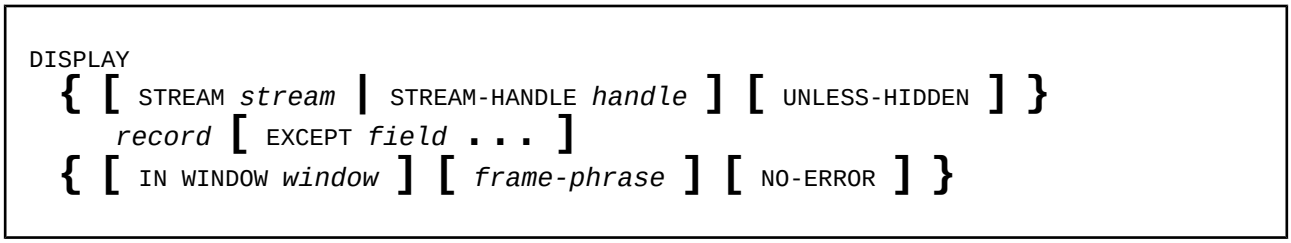
Moves data to a screen buffer and displays the data on the screen or other output destination. The AVM uses frames to display data. A frame describes how constant and variable data is arranged for display and data entry. You can let ABL construct default frames or you can explicitly describe frames and their characteristics.

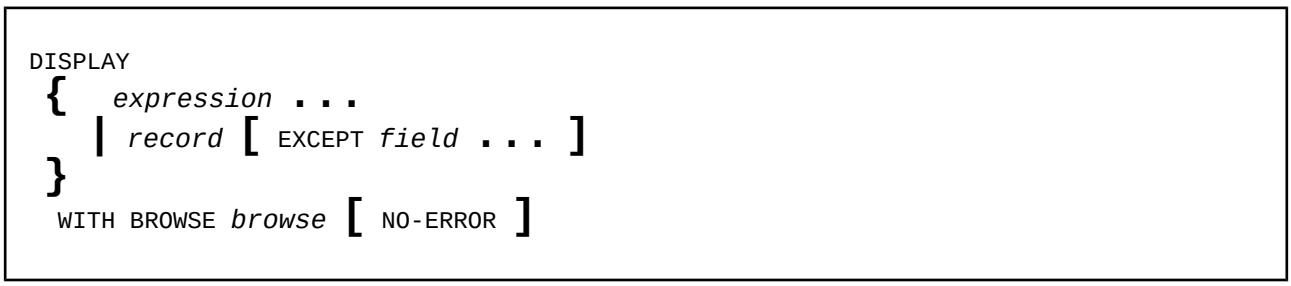
Data movement



Syntax







STREAM *stream*

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream. See the DEFINE STREAM statement reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams.

STREAM-HANDLE *handle*

Specifies the handle to a stream. If *handle* is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

UNLESS-HIDDEN

Restricts DISPLAY to fields whose HIDDEN attribute is FALSE.

expression

Identifies a constant, field name, variable name, or expression that results in the value you want to display. This can also be the built-in field name, proc-text, that returns a character string of column values from a row returned by a stored procedure proc-text-buffer.

If *expression* is a simple field or variable, the AVM checks to see if that particular field or variable is used previously in the same frame. If it has, the AVM displays the field or variable in the same frame field as the earlier instance of that field or variable.

In array fields, array elements with constant subscripts are treated just as any other field. Array fields with no subscripts are expanded as though you had typed in the implicit elements.

Note: You cannot display elements of an indeterminate array parameter or variable with no size. You can use the EXTENT statement to set the number of elements in the array before displaying them. For more information, see the EXTENT statement reference entry.

Note: You cannot display the whole array if the array is indeterminate, but you can display individual elements.

If you reference `a[i]` in the same frame that you reference a or `a[constant]`, `a[i]` overlays the appropriate frame field based on the value of i. It is displayed in a new frame field for `a[i]`. For example.

r-arry.p

```
/*1*/ DEFINE VARIABLE ix AS INTEGER NO-UNDO.

/*2*/ FOR EACH SalesRep:
/*3*/   DISPLAY SalesRep.SalesRep SalesRep.Region SalesRep.MonthQuota.
/*4*/   DO ix = 1 TO 12:
/*5*/     SET SalesRep.MonthQuota[ix] WITH 1 COLUMN.
/*6*/   END.
/*7*/   DISPLAY SalesRep.MonthQuota
/*8*/     WITH FRAME a COLUMN 40 ROW 3 1 COLUMN.
/*9*/ END.
```

Here, `month-quota[i]` is referenced in the same frame that `month-quota` is referenced. That is, line 5 references `month-quota[i]` and line 3 references `month-quota`. Both references use the same frame. Therefore, instead of creating a new frame field for `month-quota[i]`, the AVM uses the same frame fields created for the entire `month-quota` array.

In the next procedure, line 4 references only elements 1 and 2. Therefore, when the AVM tries to overlay `month-quota[i]` in line 6, there is only room for elements 1 and 2. The AVM returns an error after you enter data for those two elements.

r-arry2.p

```
/*1*/ DEFINE VARIABLE ix AS INTEGER NO-UNDO.

/*2*/ FOR EACH SalesRep:
/*3*/   DISPLAY SalesRep.SalesRep SalesRep.Region
/*4*/     SalesRep.MonthQuota[1] SalesRep.MonthQuota[2].
/*5*/   DO ix = 1 TO 12:
/*6*/     SET month-quota[ix] WITH 1 COLUMN.
/*7*/   END.
/*8*/   DISPLAY month-quota WITH FRAME a COLUMN 40 ROW 3 1 COLUMN.
/*9*/ END.
```

The following example shows a solution to that problem:

r-arry3.p

```
/*1*/ DEFINE VARIABLE ix AS INTEGER NO-UNDO.

/*2*/ FOR EACH SalesRep:
/*3*/   DISPLAY SalesRep.SalesRep SalesRep.Region
/*4*/     SalesRep.MonthQuota[1] SalesRep.MonthQuota[2] WITH 6 DOWN.
/*5*/   FORM ix SalesRep.MonthQuota[ix].
/*6*/   DO ix = 1 TO 12:
/*7*/     DISPLAY ix NO-LABEL.
/*8*/     SET SalesRep.MonthQuota[ix].
/*9*/   END.
/*10*/  DISPLAY month-quota WITH FRAME a COLUMN 40 ROW 3 1 COLUMN.
/*11*/ END.
```

If you explicitly reference `a[ix]` in a `FORM` statement, regular array fields (`month-quota[1]` and `month-quota[2]` in this example) are not overlaid.

`format-phrase`

Specifies one or more frame attributes for a field, variable, or expression. For more information on *`format-phrase`*, see the Format phrase reference entry.

`aggregate-phrase`

Identifies one or more aggregate values to be calculated optionally based on a change in a break group. This is the syntax for *`aggregate-phrase`*:

```
{   AVERAGE
  | COUNT
  | MAXIMUM
  | MINIMUM
  | TOTAL
  | SUB-AVERAGE
  | SUB-COUNT
  | SUB-MAXIMUM
  | SUB-MINIMUM
  | SUB-TOTAL
} ... [ LABEL aggr-label ] [ BY break-group ] ...
```

For more information on *`aggregate-phrase`*, see the Aggregate phrase reference entry.

`WHEN` *`expression`*

Displays an item only when the expression used in the `WHEN` option has a value of TRUE. Here, *`expression`* is a field name, variable name, or expression whose value is logical.

`@` *`base-field`*

The *`base-field`* must be the name of a field or variable; it cannot be an expression or constant. The field or variable must be viewed as a fill-in or text widget on the display.

The AVM reserves enough space for the *base-field* to hold the longest format displayed there. All right-justified fields (numerics that do not use side labels) are right justified within the reserved area. The label is left or right justified according to the *base-field*. Whenever you enter data into the *base-field*, the AVM blanks out any characters to the left or right of the area used by the field being displayed.

The AVM underlines a screen area that is the longer of the *base-field* and the overlaying field. However, you can enter as many characters as there are spaces in the format of the field.

To determine the format to use for displaying the expression at the *base-field*, the AVM looks at the following and uses the first format that applies:

- An explicit FORMAT phrase used with the *expression*.
- If the expression is a character string constant, a format that accommodates that string.
- If the data type of the expression matches that of the *base-field*, the format of the *base-field*.
- The standard format of the expression as if it were displayed without a *base-field*.

SPACE [(*n*)]

Identifies the number (*n*) of blank spaces the AVM inserts after the displayed expression displays. The *n* can be 0. If the number of spaces is more than the spaces left on the current line of the frame, the AVM starts a new line and discards extra spaces. If you do not use this option or do not use *n*, the AVM inserts one space between items in the frame.

SKIP [(*n*)]

Identifies the number (*n*) of blank lines the AVM needs to insert after the expression is displayed. The *n* can be 0. If you do not use this option, the AVM does not skip a line between expressions unless the expressions do not fit on one line. If you use the SKIP option but do not specify *n,* or if *n* is 0, the AVM starts a new line unless it is already at the beginning of a new line.

IN WINDOW *window*

Identifies the window where the expression is displayed. The expression *window* must evaluate to the handle of a window.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

record

Identifies the name of the record you want to display. Naming a record is shorthand for listing each field individually. This can also be the built-in buffer name, proc-text-buffer, that returns each row retrieved by a stored procedure.

To display a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

EXCEPT field . . .

Indicates that the AVM displays all fields except those fields listed in the EXCEPT phrase.

WITH BROWSE browse

Indicates that the AVM displays the values into the current row of the specified browse widget.

Note: DISPLAY . . . WITH BROWSE cannot be used with a dynamic browse. Instead, the user must set the browse column's SCREEN-VALUE attributes.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

The DISPLAY statement, with NO-ERROR in effect, may continue to execute even if it encounters an error resolving some part of an expression. This attempt to complete the operation may result in displaying the empty string at the location of the intended output.

Example

This procedure generates a hierarchical report of Customers (sorted by state and name), the orders belonging to those Customers, and the order-lines belonging to each order:

r-disp.p

```
FOR EACH Customer NO-LOCK BY Customer.State BY Customer.Name:
  DISPLAY Customer.State Customer.CustNum Customer.Name.
  FOR EACH Order OF Customer NO-LOCK:
    DISPLAY Order.OrderNum Order.Name Order.ShipDate Order.PromiseDate.
    FOR EACH OrderLine OF Order NO-LOCK, Item OF OrderLine NO-LOCK:
      DISPLAY OrderLine.LineNum Item.ItemName OrderLine.Qty
        OrderLline.price.
    END.
  END.
END.
```

This procedure lists each order, Customer information, and the OrderLines for each Order. The procedure calculates an Order-value for each of the OrderLines of an Order, and adds those values to produce a total value for an entire Order.

r-disp2.p

```
FOR EACH Order NO-LOCK, Customer OF Order NO-LOCK:
  DISPLAY Order.OrderNum Customer.Name Order.ShipDate Order.PromiseDate.
  FOR EACH OrderLine OF Order NO-LOCK, Item OF OrderLine NO-UNDO:
    DISPLAY OrderLine.LineNum Item.ItemName OrderLine.Qty OrderLine.Price
      OrderLine.Qty * OrderLine.Price (TOTAL) LABEL "Order-value".
  END.
END.
```

The r-disp3.p procedure displays a name and address list in a mailing label. The SKIP and FORMAT options are used to produce a standard address format. The WHEN option suppresses the display of the PostalCode field if there is no postal code value in the field.

r-disp3.p

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.Name SKIP Customer.Address SKIP Customer.Address2 SKIP
    Customer.City + ", " + Customer.State FORMAT "x(16)"
    Customer.PostalCode WHEN Customer.PostalCode NE "" SKIP(2)
    WITH NO-BOX NO-LABELS USE-TEXT.
END.
```

Notes

- When ABL compiles a procedure, it uses a top-to-bottom pass of the procedure to design all the frames that procedure requires, adding field and related format attributes as it goes through the procedure.
- If you are displaying data that contains special control characters such as tabs, form feeds, or backspaces, be sure to use an EDITOR widget of the appropriate size for *expression* or *base-field*, or use the VIEW-AS EDITOR option from *format-phrase* in the DISPLAY statement. Otherwise, do not display data containing these characters.
- If you use a single qualified identifier with the DISPLAY statement, the Compiler first interprets the reference as *dbname.table-name*. If the Compiler cannot resolve the reference as *dbname.table-name*, it tries to resolve it as *table-name.fieldname.*
- If you invoke the DISPLAY statement for a frame, the AVM brings the frame into view unless the HIDDEN attribute for the frame or one of its ancestor frames or windows is TRUE.
- For more information on using the built-in field and buffer names, proc-text and proc-text-buffer in a DISPLAY statement, see the OpenEdge DataServer Guides (*Use the Microsoft SQL Data Server* and *Use the Oracle Data Server*).
- You cannot use the DISPLAY statement to display an object reference for a class instance directly. To display an object reference, you must first convert it using the INTEGER, INT64, or STRING function and display the result.

See also

ACCUM function, Aggregate phrase, DEFINE BROWSE statement, DEFINE FRAME statement, DOWN statement, EXPORT statement, FORM statement, Format phrase, Frame phrase, MESSAGE statement, NO-ERROR option, PAGE statement, PUT statement, PUT SCREEN statement, Stream object handle, UP statement, VIEW-AS phrase

DO statement

Groups statements into a single block, optionally specifying processing services or block properties. Use an END statement to end a DO block.

Syntax

This is the syntax for a DO block:

```
[ label : ]
DO
  { [ FOR record [ , record ] ... ] }
  [ preselect-phrase ]
  [ query-tuning-phrase ]
  [ variable = expression1 TO expression2 [ BY k ] ]
  [ WHILE expression ]
  [ TRANSACTION ]
  [ stop-after-phrase ]
  [ on-endkey-phrase ]
  [ on-error-phrase ]
  [ on-quit-phrase ]
  [ on-stop-phrase ]
  { [ frame-phrase ] } :
  <block of statements>
END.
```

FOR *record*[, *record*]...

Names the buffer you want to work with in the block and scopes the buffer to the block. The scope of a record determines when the buffer for that record is cleared and written back to the database. See *BL Essentials* for more information on record scoping.

To work with a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See Record phrase for more information.

preselect-phrase

The PRESELECT phrase finds selected records from one or more tables. You can access those preselected records with statements such as FIND NEXT.

```
PRESELECT
  [ EACH | FIRST | LAST ]record-phrase
  [ , [ EACH | FIRST | LAST ] record-phrase ]...
  [
    [ BREAK ]
    { BY expression[ DESCENDING ]}...
  ]
```

For more information, see the PRESELECT phrase.

query-tuning-phrase

Allows programmatic control over the execution of a DataServer query.

```
QUERY-TUNING (
  {
    [ BIND-WHERE | NO-BIND-WHERE ]
    [ CACHE-SIZE integer ]
    [ DEBUG { SQL | EXTENDED } | NO-DEBUG ]
    [ INDEX-HINT | NO-INDEX-HINT ]
    [ JOIN-BY-SQLDB | NO-JOIN-BY-SQLDB ]
    [ LOOKAHEAD | NO-LOOKAHEAD ]
    [ SEPARATE-CONNECTION | NO-SEPARATE-CONNECTION ]
  }
  )
```

For more information, see the OpenEdge DataServer Guides (*Use the Microsoft SQL Data Server* and *Use the Oracle Data Server*).

variable = *expression1* TO *expression2* [BY *k*]

The name of a field or variable whose value is incremented in a loop. *expression1* is the starting value for *variable* on the first iteration of the loop. *k* is the amount to add to *variable*after each iteration; it must be a constant. *k* defaults to 1 if not specified. variable, expression1 and *expression2* must be integers.

When *variable* exceeds *expression2* (or is less than *expression2* if *k*is negative) the loop ends. Since *expression1* is compared to *expression2* at the start of the first iteration of the block, the block can be executed zero times. *expression2* is re-evaluated on each iteration of the block.

WHILE *expression*

Indicates that the DO block continues processing the statements within it. Using the WHILE option turns a DO block into an iterating block. The block iterates as long as the condition specified by the expression is TRUE. The expression is any combination of constants, operators, field names, and variable names that yield a logical value.

TRANSACTION

Identifies the DO block as a system transaction block. The AVM starts a system transaction for each iteration of a transaction block if there is not already an active system transaction. See *ABL Essentials* for more information on transactions.

stop-after-phrase

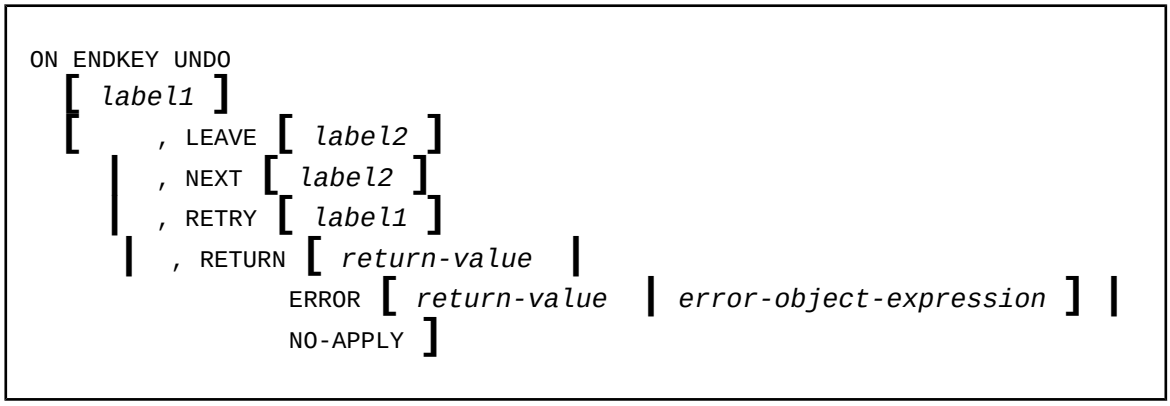
Specifies a time-out value (in seconds) for the block. This is the syntax for the STOP-AFTER phrase:

```
STOP-AFTER time-limit
```

For more information see STOP-AFTER phrase.

on-endkey-phrase

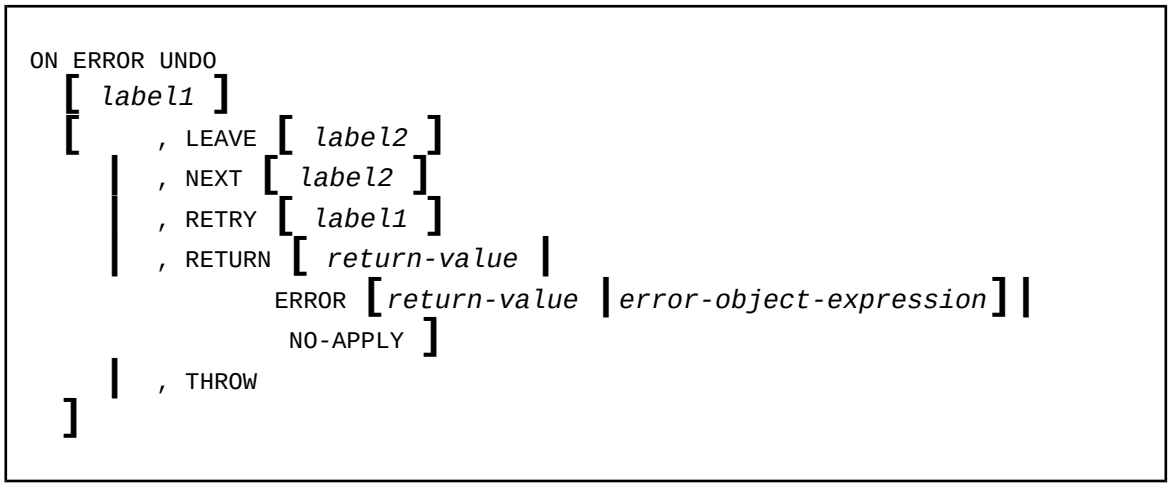
Describes the processing that takes place when the **ENDKEY** condition occurs during a block. This is the syntax for the ON ENDKEY phrase:



For more information, see the ON ENDKEY phrase.

on-error-phrase

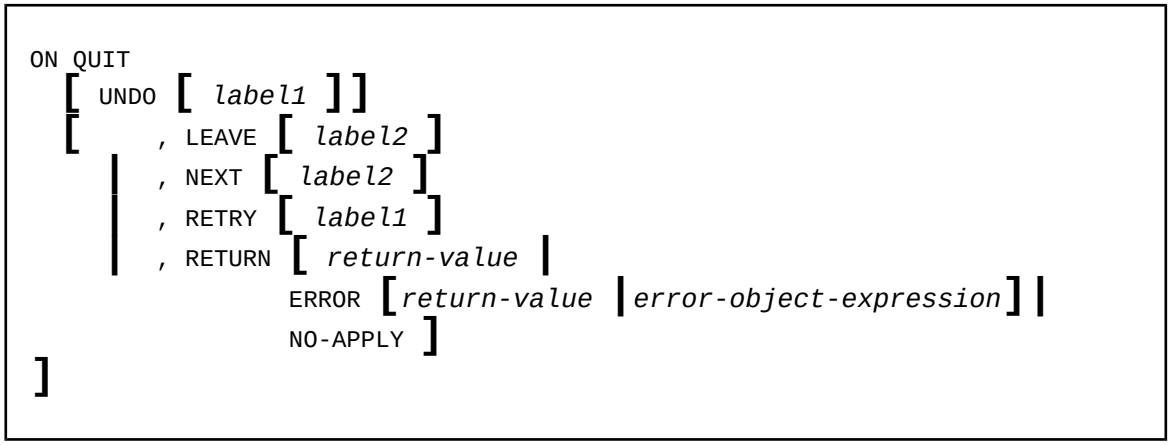
Describes the processing that takes place when there is an error during a block. This is the syntax for ON ERROR phrase:



For more information, see the ON ERROR phrase.

on-quit-phrase

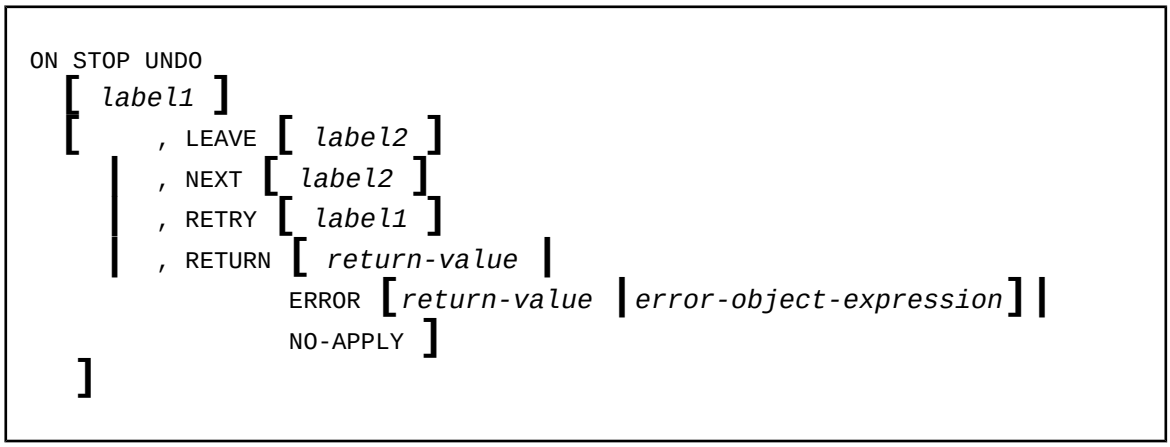
Describes the processing that takes place when a QUIT statement is executed during a block. This is the syntax for the ON QUIT phrase:



For more information, see the ON QUIT phrase.

on-stop-phrase

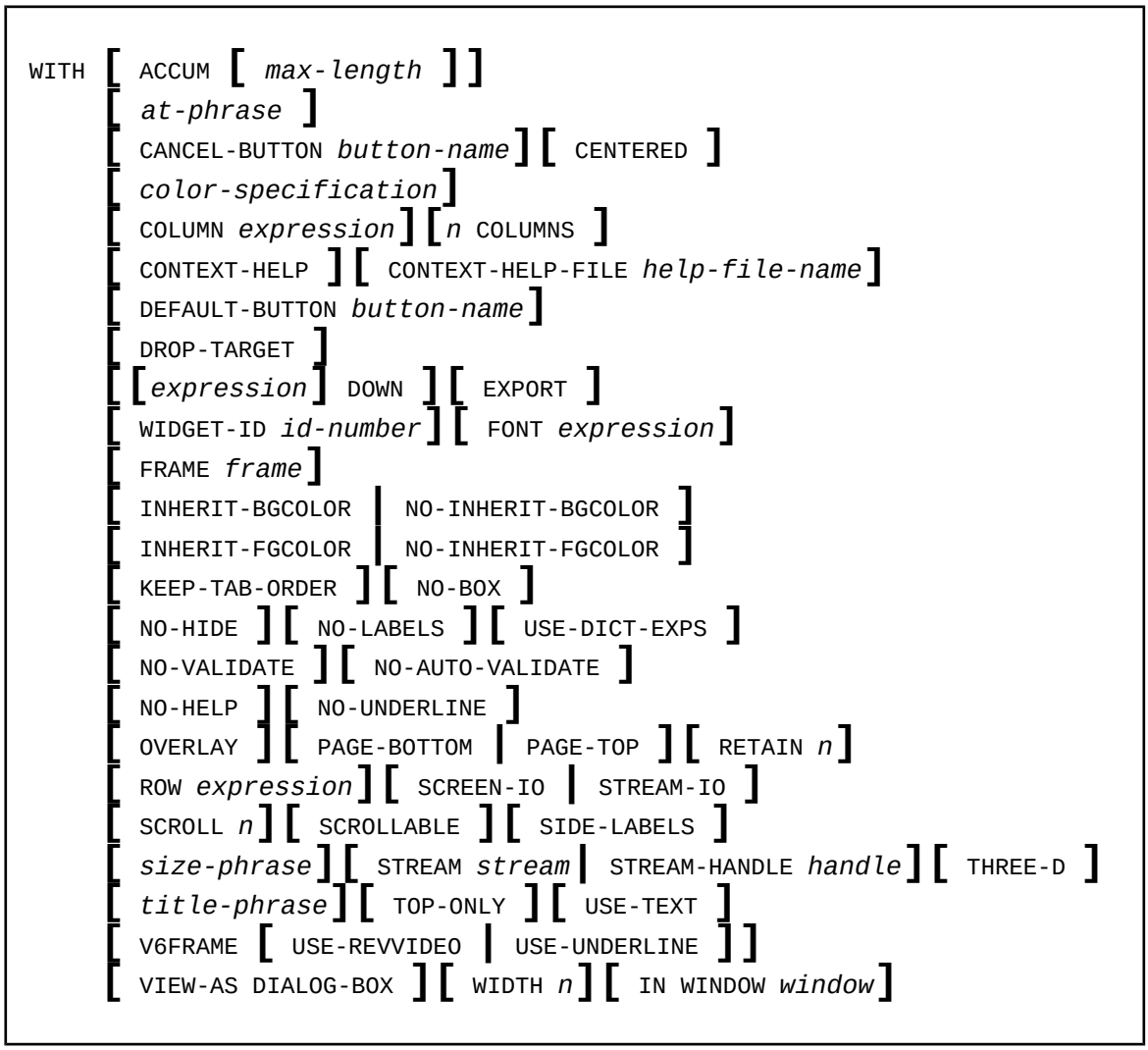
Describes the processing that takes place when the STOP condition occurs during a block. This is the syntax for the ON STOP phrase:



For more information, see the ON STOP phrase.

frame-phrase

Specifies the overall layout and processing properties of a frame. This is the syntax for the frame phrase:



For more information, see the Frame phrase.

Example

This procedure goes through the `Customer` table and reduces the `CreditLimit` to 80000 for those Customers that exceed this amount. The procedure uses a simple `DO` block to process two statements if the `CreditLimit` is over 80000. Simple `DO` blocks are most useful for grouping statements together for conditional situations, such as `IF...THEN...ELSE`.

r-do.p

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.Name Customer.CreditLimit.
  PAUSE 3.
  IF Customer.CreditLimit > 80000 THEN DO:
    Customer.CreditLimit = 80000.
    DISPLAY Customer.Name Customer.CreditLimit.
  END.
END.
```

Notes

- Use a DO statement rather than a REPEAT statement when you loop through elements of an array. This is more efficient since the AVM does not create separate subtransactions within the transaction.

  The first example, which uses DO, is more efficient than the second:

  ```
  DO ix = 1 TO 12:
    SalesRep.MonthQuota[ix] = 0.
  END.
  ```

  ```
  REPEAT ix = 1 TO 12:
    SalesRep.MonthQuota[ix] = 0.
  END.
  ```

- For SpeedScript, the *on-endkey-phrase* and the *on-quit-phrase* do not apply.

See also

FIND statement, FOR statement, Frame phrase, ON ENDKEY phrase, ON ERROR phrase, ON QUIT phrase, ON STOP phrase, Record phrase, REPEAT statement, STOP-AFTER phrase

DOS statement
(Windows only)

Runs a program, DOS command, or DOS batch file, or starts the DOS command processor, which allows interactive processing of DOS commands.

Syntax

```
DOS [ SILENT ] [ command-token | VALUE ( expression ) ] ...
```

SILENT

After processing a DOS statement, the AVM shell pauses, and prompts you to press **SPACEBAR** to continue. When you press **SPACEBAR**, the AVM clears the window and continues processing. You

can use the SILENT option to eliminate this pause. Use this option only if you are sure that the DOS program, command, or batch file will not generate output to the window.

command-token | VALUE (*expression*)

One or more command (*command-token*) words and symbols that you want to pass to a DOS command processor. The VALUE option generates the command tokens included in *expression*, a character string expression. The specified combination of *command-token* and VALUE (*expression*) options can form any legal combination of commands and command options permitted by the DOS command processor, including programs, DOS commands, and batch files. If you do not use any of these options, the DOS statement invokes the DOS command processor, which remains until you exit it.

The commands can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

Example

On UNIX, this procedure runs the UNIX `ls` command. In Windows, this procedure runs the DOS `dir` command. On other platforms, the AVM displays a message stating that the operating system is unsupported.

r-dos.p

```
IF OPSYS = "UNIX" THEN UNIX ls.
ELSE IF OPSYS = "WIN32" THEN DOS dir.
ELSE DISPLAY OPSYS "is an unsupported operating system".
```

Note

If you use the DOS statement in a procedure and the procedure compiles on a UNIX system, the procedure runs, as long as flow of control does not pass through the DOS statement while running on UNIX. Use the OPSYS function to return the name of the operating system where a procedure is being run. This function lets you write applications that are portable among OpenEdge-supported operating systems even if they use the DOS, UNIX, etc. statements.

See also

{ } Preprocessor name reference, OPSYS function, UNIX statement

DOWN statement

Positions the cursor on a new line in a down or multi-line frame.

When the block specifying the down frame iterates, the AVM automatically advances one frame line. Use the DOWN statement if you want to move to a different display line at any time.

For more information on down frames, see the DOWN option of the Frame phrase.

Note: Does not apply to SpeedScript programming.

Syntax

```
DOWN [ STREAM stream | STREAM-HANDLE handle ] [ expression ]
     { [frame-phrase ] }
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream. See the DEFINE STREAM statement reference entry and the "Alternate I/O Sources" topics in *OpenEdge Programming Interfaces* for more information on streams.

STREAM-HANDLE *handle*

Specifies the handle to a stream. If *handle* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

expression

The number of occurrences of data in the frame that you want to move down.

DOWN is the same as DOWN 1, except for the following:

- Nothing happens until a data handling statement affects the screen.
- Several DOWN statements in a row with no intervening displays are treated as a single DOWN 1.

DOWN 0 does nothing. If *n* is negative, the result is the same as UP ABS(*n*).

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

Example

This procedure prints a Customer report that is sorted by State, with one line after the last Customer in each state:

r-down.p

```
DEFINE VARIABLE laststate AS CHARACTER NO-UNDO.

FOR EACH Customer NO-LOCK BY Customer.State:
  IF Customer.State <> laststate THEN DO:
    IF laststate <> "" THEN DOWN 1.
    laststate = Customer.State.
  END.
  DISPLAY Customer.CustNum Customer.Name Customer.City Customer.State.
END.
```

Notes

- After displaying a down frame, the AVM automatically advances to the next frame line on each iteration of the block where the frame belongs. This is true whether or not you use the DOWN statement. If you do not want the AVM to advance automatically, name the frame outside of the block involved (the statement FORM WITH FRAME *frame* names a frame and scopes that frame to the higher block).
- When the AVM reaches the last frame line and encounters a DOWN statement, it clears the frame and starts at the top line of the frame, unless you used the SCROLL option on the frame. In that case, the AVM scrolls the frame up one iteration only, to make room for the next iteration.

See also

DEFINE STREAM statement, Frame phrase, SCROLL statement, Stream object handle, UP statement

---

DYNAMIC-CAST function

Returns a new object reference to the same class instance as an existing object reference, but with a different data type. This different data type is cast from the object type of the original object reference according to another object type specified by a run-time expression. The object types must be related, where one is a class type and the other is subclass of that class type or where one is an interface type and the other is a class that implements the interface of that type.

When you cast an object reference, ABL treats it as if it referenced an instance of the object type to which it is cast. The underlying class hierarchy of the object instance does not change.

Syntax

```
DYNAMIC-CAST( object-reference , expression ).
```

object-reference

An object reference defined with the object type to be cast.

expression

A character expression that evaluates to the fully qualified type name for the ABL or .NET class or interface type to which the object reference is cast. This object type must be a class type in a class hierarchy that includes or implements the *object-reference* data type or it must be an interface type that the *object-reference* data type implements. Specify an object type name using the syntax described in the Type-name syntax reference entry. Any present USING statement has no effect. If no package is specified, the class or interface name must represent the complete type name.

Notes

- You typically cast an object reference down a class hierarchy—that is, from a super class to a derived class within a class hierarchy, or from an interface to a class that implements that interface. However, you do not always need to explicitly cast an object reference. Because a derived class contains all the super classes in its inherited class hierarchy, ABL implicitly casts any object reference up within its class hierarchy, and because a class that implements an interface implements all of the methods specified for the interface, ABL implicitly casts any object reference from an implementing class to any interface that the class implements.

- At run time, ABL verifies that the object type specified by *expression* is within the class hierarchy of the specified object reference. Therefore, if you access a class member on the cast object reference that exists for the cast data type, but the referenced object does not actually define the accessed class member, the AVM raises a run-time ERROR.
- A .NET generic type can be part of a cast. For example, you can cast from a `System.Object` to a `"System.Collections.Generic.List<SHORT>"`, because all .NET classes, including generic classes, derive from the .NET root class. However, note that you cannot cast from a `"System.Collections.Generic.List<System.Object>"` to a `"System.Collections.Generic.List<System.Windows.Forms.Button>"`. You cannot assign a `"List<Button>"` reference to an object reference defined as a `"List<Object>"`, because, even though the type parameters are compatible, the two objects as a whole are not equivalent and have no inheritance relationship. Therefore, a cast between these two objects cannot work either. For more information on .NET generic types, see the Data types reference entry.
- You can also use the CAST function to perform all casting operations at compile time. The primary reason for using the DYNAMIC-CAST function is to cast object references based on run-time conditions that determine the object type to use for the cast. Such conditions are common in applications that conform to the OpenEdge Reference Architecture (OERA). For more information on the OERA, see Progress Communities: https://community.progress.com/.
- You can use the DYNAMIC-CAST function to cast a parameter in a parameter list for a method using the following syntax, where *expression* evaluates to a subclass of the *object-reference* type:

```
method-name( INPUT DYNAMIC-CAST( object-reference, expression ), ... ).
```

- You can use the DYNAMIC-CAST function to cast a temp-table field, which is defined as a `Progress.Lang.Object`, to use as an object of another class type. For example:

```
DEFINE VARIABLE RCustObj AS CLASS acme.myObjs.CustObj.

DEFINE TEMP-TABLE mytt NO-UNDO
  FIELD CustObj AS Progress.Lang.Object.

RCustObj = DYNAMIC-CAST(mytt.CustObj, "acme.myObjs.CustObj").
```

  You can now use the object reference in `RCustObj` to invoke methods in the `acme.myObjs.CustObj` class.

- You **cannot** use the DYNAMIC-CAST function to cast an object reference to a subclass and invoke a method defined in that subclass using the following syntax:

```
DYNAMIC-CAST( object-reference, object-type-name ):method-name(
parameters ).
```

  Similarly, you cannot use this syntax to invoke a method on a class that implements the referenced interface from which you cast the specified class.

See also

CAST function, Type-name syntax, USING statement

DYNAMIC-CURRENT-VALUE function

Returns the current INT64 value of a sequence defined in the specified database.

Syntax

```
DYNAMIC-CURRENT-VALUE(
  sequence-expression , logical-dbname-expression  [ , tenant-id ] )
```

sequence-expression

A character expression that evaluates to the name of a sequence.

logical-dbname-expression

A character expression that evaluates to the name of a connected database in which the sequence is defined.

tenant-id

An integer expression that evaluates to the tenant ID of a regular tenant, including the default tenant (0). This option applies only to a multi-tenant sequence specified by *sequence-expression* and is intended for access primarily by a super-tenant user.

If you are a regular-tenant user and you specify *tenant-id*, the specified tenant must be the same as the tenant associated with the database connection identity for *logical-dbname-expression* or the AVM raises a run-time error.

If you are a super-tenant user and you do not specify *tenant-id*, the function assumes the effective tenant ID (GET-EFFECTIVE-TENANT-ID function).

If the sequence specified by *sequence-expression* is non-multi-tenant (a shared sequence in a multi-tenant database), and you specify *tenant-id*, the option is ignored.

Notes

- If *logical-dbname-expression* contains the Unknown value (?), the AVM generates a run-time error.
- The user must have CAN-READ privileges on the _Sequence table to use the DYNAMIC-CURRENT-VALUE function.
- The current value of a sequence can be one of the following:
  - The initial value specified in the Data Dictionary.
  - The last value set with either the DYNAMIC-CURRENT-VALUE statement or the DYNAMIC-NEXT-VALUE function.
  - The Unknown value (?) if the sequence has exceeded its minimum or maximum and is not cycling.
- If *sequence-expression* is a multi-tenant sequence in the database, each regular tenant has their own current value of the sequence. So, the same values are returned for each tenant that invokes this function. If the sequence is shared in a multi-tenant database, the values returned by this function are unique across all tenants in the database.

Caution: Be very careful using multi-tenant sequences when you are updating shared tables or multi-tenant tables for a tenant group. For more information, see the topics on using multi-tenant-enabled sequences in *OpenEdge Programming Interfaces*.

- Sequence values are stored in the database in which they are defined, and persist between each invocation of the DYNAMIC-CURRENT-VALUE statement or DYNAMIC-NEXT-VALUE function.
- You cannot invoke the DYNAMIC-CURRENT-VALUE function from within a WHERE clause. Doing so generates a compiler error. To use a result from the DYNAMIC-CURRENT-VALUE function in a WHERE clause, assign the result to a variable, then use the variable in the WHERE clause.
- You can use any combination of the DYNAMIC-NEXT-VALUE function, DYNAMIC-CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, and their non-dynamic versions.

See also

CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, DYNAMIC-CURRENT-VALUE statement, DYNAMIC-NEXT-VALUE function, NEXT-VALUE function

DYNAMIC-CURRENT-VALUE statement

Resets the current integer value of a sequence defined in the specified database.

Syntax

```
DYNAMIC-CURRENT-VALUE(
  sequence-exp , logical-dbname-exp [ , tenant-id ] ) = expression
```

sequence-exp

A character expression that evaluates to the name of a sequence.

logical-dbname-exp

A character expression that evaluates to the name of a connected database in which the sequence is defined.

tenant-id

An integer expression that evaluates to the tenant ID of a regular tenant, including the default tenant (0). This option applies only to a multi-tenant sequence specified by *sequence-expression* and is intended for access primarily by a super-tenant user.

If you are a regular-tenant user and you specify *tenant-id*, the specified tenant must be the same as the tenant associated with the database connection identity for *logical-dbname-expression* or the AVM raises a run-time error.

If you are a super-tenant user and you do not specify *tenant-id*, the function assumes the effective tenant ID (GET-EFFECTIVE-TENANT-ID function).

If the sequence specified by *sequence-expression* is non-multi-tenant (a shared sequence in a multi-tenant database), and you specify *tenant-id*, the option is ignored.

expression

An integer expression assigned as the current value of the specified sequence. If *expression* is outside the boundary set by the initial value (at one end) and the lower limit or upper limit (at the other end) for the sequence, the AVM returns an error, and the sequence value remains unchanged.

Note: Compound assignment operators (+=, -=, *=, /=) can be used with the DYNAMIC-CURRENT-VALUE statement. For more information, see += Addition assignment operator, -= Subtraction assignment operator, *= Multiplication assignment operator, and /= Division assignment operator.

Notes

- If *logical-dbname-exp* contains the Unknown value (?), the AVM generates a run-time error.
- You cannot set a sequence to the Unknown value (?).
- If *sequence-exp* is a multi-tenant sequence in the database, each regular tenant has their own current value of the sequence. So, each tenant that invokes this statement can set their own current values, which can be the same or different than the current values set by other tenants. If the sequence is shared in a multi-tenant database, any current value set by this statement is shared by all tenants in the database.

  Caution: Be very careful using multi-tenant sequences when you are updating shared tables or multi-tenant tables for a tenant group. For more information, see the topics on using multi-tenant-enabled sequences in *OpenEdge Programming Interfaces*.

- The user must have CAN-WRITE privileges on the _Sequence table to use the DYNAMIC-CURRENT-VALUE statement.
- The value of a sequence set by the DYNAMIC-CURRENT-VALUE statement persists in the database until the next DYNAMIC-CURRENT-VALUE statement or DYNAMIC-NEXT-VALUE function is invoked for the sequence, or until the sequence is deleted from the database.
- You can use any combination of the DYNAMIC-NEXT-VALUE function, DYNAMIC-CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, and their non-dynamic versions.

See also

CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, DYNAMIC-CURRENT-VALUE function, DYNAMIC-NEXT-VALUE function, NEXT-VALUE function

---

DYNAMIC-ENUM function

Returns an instance of the enumeration type for the specified enumeration member.

Syntax

```
[ return-value = ] DYNAMIC-ENUM( enum-type-name, enum-member )
```

return-value

Specifies a data element that is assigned the returned value. The AVM checks that the enum instance returned can be assigned to the target, and the AVM raises an error if the data types aren't compatible.

enum-type-name

Specifies the name of an ABL or .NET enumeration type that defines the member. This is a CHARACTER expression that the AVM evaluates to the type name of an enumeration type at runtime.

enum-member

Specifies an enum member in one of two ways:

- A CHARACTER expression that evaluates to the member name at runtime. For flag enums, this can be a comma-delimited list of member names for two or more members defined by the enum type.
- An INTEGER or INT64 expression that evaluates to the underlying numeric value of the member at runtime. For flag enums, this numeric value can represent two or more members defined by the enum type.

Note: The function generates a runtime error if any specified character or numeric value does not correspond to a defined member.

Example

In the following code fragment, `Direction` is an enum with a member defined for each of the four cardinal directions (see the ENUM statement for an example of how `Direction` could be defined), and `cNeededDirection` is a string that evaluates to the name of one of the members defined by `Direction`. The code then uses DYAMIC-ENUM to set `myDirection` as an object reference to an instance of the member specified by `cNeededDirection`:

```
DEFINE INPUT PARAMETER cNeededDirection AS CHARACTER.
DEFINE VARIABLE myDirection AS Direction.

myDirection = DYNAMIC-ENUM("Direction", cNeededDirection).
```

See also

ENUM statement

DYNAMIC-FUNCTION function

Invokes a user-defined function. The AVM evaluates the name of the function (and the procedure handle, if any) at run time.

Syntax

```
DYNAMIC-FUNCTION
  ( function-name [ IN proc-handle ]
    [ , param1 [ , param2 ] ... ]
  )
```

function-name

A CHARACTER expression that returns the name of a user-defined function. The AVM evaluates *function-name* at run time.

IN *proc-handle*

An expression that returns a handle to the procedure that defines the function. The AVM evaluates *proc-handle* at run time.

param1, param2, ...

Parameters of the user-defined function. You must supply names of actual data items—actual parameter names—not CHARACTER expressions that return parameter names.

Note: ABL cannot check the mode and type of the parameters at compile time, since the AVM does not evaluate *function-name* until run time.

Example

The following procedure demonstrates the DYNAMIC-FUNCTION function:

r-funfun.p

```
/* Requires a connection to the Sports2020 database. Define data items */
DEFINE VARIABLE funcs   AS CHARACTER NO-UNDO EXTENT 5
  INITIAL ["firstrec","lastrec","nextrec","prevrec","quitting"].
DEFINE VARIABLE action  AS CHARACTER NO-UNDO LABEL "Action" FORMAT "x"
  INITIAL "N".
DEFINE VARIABLE idx     AS INTEGER   NO-UNDO.
DEFINE VARIABLE alldone AS LOGICAL   NO-UNDO.

FORM WITH FRAME x SIDE-LABELS 2 COLUMNS 1 DOWN COLUMN 25.

/* Define user-defined functions */
FUNCTION dispcust RETURNS LOGICAL:
  DISPLAY Customer EXCEPT Customer.Comments WITH FRAME x.
END.

FUNCTION firstrec RETURNS LOGICAL:
  FIND FIRST Customer.
  dispcust().
  RETURN yes.
END.

FUNCTION lastrec RETURNS LOGICAL:
  FIND LAST Customer.
  dispcust().
  RETURN yes.
END.
```

```
FUNCTION nextrec RETURNS LOGICAL:
  FIND NEXT Customer NO-ERROR.
  IF AVAILABLE Customer THEN
     dispcust().
  RETURN AVAILABLE(Customer).
END.

FUNCTION prevrec RETURNS LOGICAL:
  FIND PREV Customer NO-ERROR.
  IF AVAILABLE Customer THEN
     dispcust().
  RETURN AVAILABLE(Customer).
END.

FUNCTION quitting RETURNS LOGICAL:
  alldone = yes.
  RETURN no.
END

/* Main routine */
REPEAT WHILE NOT alldone:
  UPDATE action HELP
    "Enter F(irst), L(ast), N(ext), P(rior), or Q(uit) to navigate.".
  idx = LOOKUP(action,"f,l,n,p,q").
  IF idx EQ 0 THEN DO:
    MESSAGE "Enter F(irst), L(ast), N(ext), P(rior), or Q(uit)"
      VIEW-AS ALERT-BOX.
    NEXT.
  END.
  DISPLAY DYNAMIC-FUNCTION(funcs[idx]) LABEL "Record Found?".
END.
```

See also

FUNCTION statement

DYNAMIC-INVOKE function

Invokes a class-based method whose name is specified by a run-time expression, but whose parameters are defined at compile time.

Syntax

```
[ return-value = ] DYNAMIC-INVOKE( {class-type-name | object-reference}
  , method-name [ , parameter [ , parameter ] ... ] )
```

return-value

An optional data element that is assigned the return value from the invoked, non-void method. The AVM checks *return-value* at run time for data type compatibility with what is actually returned by the method. ABL raises a run-time error if you invoke a void method where a return value is expected.

class-type-name

The name of an ABL or .NET class type that defines the specified method as a static member. If the method is static, this parameter is a CHARACTER expression that the AVM evaluates to the type name of the class at run time. This expression must specify a class type name as described in the Type-name syntax reference entry, except that you must always specify the complete type name; any present USING statement has no effect.You cannot use *class-type-name* to call an instance method. For more information on specifying class type names, see the Type-name syntax reference entry.

object-reference

Specifies a reference to an ABL or .NET class instance that defines the specified method as an instance member. You cannot use *object-reference* to call a static method. For information on specifying object references, see the reference entry for a Class-based object reference.

method-name

A CHARACTER expression that evaluates to the method name. The AVM evaluates *method-name* at run time.

[, *parameter* **[** , *parameter* **]...** **]**

Specifies zero or more parameters passed to the method. You must provide the parameters identified by the specified method, matched with respect to number, data type, and mode. To invoke a method that is overloaded in the class, you must specify sufficient information for each *parameter* in order to disambiguate methods that have similar parameter lists. Otherwise, the AVM raises an error identifying the ambiguity.

If the parameter mode (INPUT, OUTPUT, INPUT-OUTPUT) is not specified, the mode defaults to INPUT. This differs from normal invocation of a method, where the parameter mode defaults to the method parameter. In this case, the compiler does not know the method at compile time, and therefore must use INPUT for the default, instead of inferring it from the called method. We recommended that you always supply the parameter mode.

For more information on parameter passing syntax and on disambiguating overloaded ABL and .NET methods, see the Parameter passing syntax reference entry.

Example

In this example, the `MultiplyTheValue` method in the class instance referenced by `rObj` expects a `DECIMAL` value.

```
class MyObject:
    method public decimal MultiplyTheValue(dValue as decimal):
        return dValue * 2.
    end.
end.

var decimal retValue.
var MyObject rObj = new MyObject().

retValue = dynamic-invoke(rObj, "MultiplyTheValue", 12.3).
```

Notes

- The `Invoke( )` method of the `Progress.Lang.Class` class provides similar functionality to the `DYNAMIC-INVOKE` function. The difference is that `DYNAMIC-INVOKE` has a fixed, compile-time parameter list and, therefore, does not require the creation of a `ParameterList` object at run time.
- `DYNAMIC-INVOKE` must be executed as a stand-alone statement (without assigning it to a *`return-value`*) if the method it invokes is VOID.

See also

DYNAMIC-NEW statement, Invoke( ) method (Class), New( ) method, Parameter passing syntax, Progress.Lang.Class class

DYNAMIC-NEW statement

Creates an instance of a class (object) whose class type is specified by a run-time expression, and assigns its object reference to an appropriately defined ABL data element. Once assigned, you can use the object reference to access this class instance and its PUBLIC data members, properties, and methods. For more information on object references, see the reference entry for a Class-based object reference.

Syntax

```
object-reference = DYNAMIC-NEW expression
  ( [ parameter [ , parameter ]... ] ) [ NO-ERROR ]
```

`object-reference`

The name of an ABL data element to which you want to assign the object reference of a new instance of the class specified by *`expression`*. This data element must be defined as a compatible class or interface type, and can be one of the following:

- A temp-table field defined as a Progress.Lang.Object class type using the DEFINE TEMP-TABLE statement
- A variable scoped to the current procedure, user-defined function, or method of a class, or an accessible class-based variable data member, each defined as a class or interface type using the VAR statement or DEFINE VARIABLE statement
- A parameter defined as a class or interface type for the current procedure, user-defined function, or method of a class using an appropriate DEFINE PARAMETER statement or Parameter definition syntax
- An accessible and writable class-based property defined as a class or interface type using the DEFINE PROPERTY statement

To be compatible, the data type of *`object-reference`* must be:

- The same class as the class specified by *`expression`*
- A super class of the class specified by *`expression`*
- An interface that is implemented by the class specified by *`expression`*

expression

A character expression that evaluates to a fully qualified class type name for the ABL or .NET class you want to instantiate. This expression must specify a class type name as described in the Type-name syntax reference entry, except that you must always specify the complete type name; any present USING statement has no effect. If no package (or namespace for a .NET class) is specified, the class name must represent the complete type name.

This expression cannot evaluate to:

- An ABL built-in class type name, such as `Progress.Lang.Object`
- The type name of an interface or abstract class

(**[** *parameter* **[** , *parameter* **]...** **]**)

Specifies zero or more parameters passed to a PUBLIC instance constructor that is defined for the class. You must provide the parameters identified by the specified constructor, matched with respect to number, data type, and mode.

For information on the parameter passing syntax, see the Parameter passing syntax reference entry.

For information on defining a constructor for a class, see the CONSTRUCTOR statement reference entry.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

For the DYNAMIC-NEW statement with NO-ERROR, if ERROR is raised, then the *object-reference* remains unchanged.

Example

The following contrived (non-compiling) procedure fragment shows the instantiation of a new class type specified with a variable:

```
/* Can be set to a subclass type name */
DEFINE INPUT PARAMETER myBusObjParm AS CHARACTER NO-UNDO.

/* Procedure only knows about the base class */
DEFINE VARIABLE myBusObj AS CLASS acme.myObjs.BusObj NO-UNDO.

myBusObj = DYNAMIC-NEW myBusObjParm ( ). /* Create the passed subclass */
 myBusObj:getData( ).                    /* Invoke base class method
                                             polymorphically on subclass */
```

In this case, the procedure assumes that it is operating on a class, `acme.myObjs.BusObj`, that is the base class for several subclasses, each of which implements the same set of operations for its own purposes, such as the `getData( )` method shown. When the procedure is called, its INPUT parameter is passed a character string that evaluates to a subclass type name, such as `"acme.myObjs.CustObj"`, which it then uses to instantiate a class of that type using the DYNAMIC-NEW statement. Thus, the same procedure can be called to instantiate and operate on different subclasses of the same base class, as determined by run-time conditions.

Notes

- Unlike NEW, you cannot use DYNAMIC-NEW as a function in an expression. You can use it only as a statement.

- After the assignment, *object-reference* contains a copy of the object reference value returned by DYNAMIC-NEW, which points to the same object instance, not a copy of the object created by DYNAMIC-NEW.
- Although you can assign an object reference to a temp-table field defined as a Progress.Lang.Object class type, you cannot assign an object reference to a field in a database table. For more information, see *Develop Object-oriented ABL Applications*.
- The ABL Virtual Machine (AVM) automatically deletes (garbage collects) any class instance that you create with this statement some time after no reference to that object exists in the ABL session. However, you can force any class instance to be deleted immediately by using the DELETE OBJECT statement. For more information on garbage collection for class instances, see the DELETE OBJECT statement reference entry.
- If *expression* specifies a .NET object, note that in ABL you cannot instantiate the following .NET classes:
  - Any .NET class that is defined in the default namespace, that is, where the class name is the complete object type name
  - System.Threading.Thread or any class derived from it
  - System.Delegate or any delegate type derived from it
- If *expression* specifies a GUI or non-GUI .NET object type, you **can** instantiate it within a non-GUI ABL session on Windows, including a:
  - Character mode (CHUI) client
  - Batch-mode client
  - Progress Application Server (PAS) for OpenEdge instance

  However, you **cannot** block for any .NET object events or visualize any GUI objects in a non-GUI ABL session.
- This statement can raise errors during the execution of constructors for the class being instantiated, or for any class in its inherited class hierarchy. For example:
  - A constructor in the class hierarchy executes the RETURN statement with the ERROR option or the UNDO statement with the THROW or RETURN ERROR options.
  - The class definition file for the class, a super class, or an interface could not be found.
  - The run-time parameters of the constructor for the class, or a constructor for a class in the inherited class hierarchy, are not compatible.

  When the AVM encounters one of these errors, and the constructor cannot create the class instance or its inherited class hierarchy, the AVM automatically invokes the destructor for any class that has already been constructed while building the class hierarchy for the object.

  For more information on errors raised by instantiating classes, see *Develop Object-oriented ABL Applications*.
- The New( ) method of the Progress.Lang.Class class provides similar functionality to the DYNAMIC-NEW function. The advantage to the latter is that it has a fixed, compile-time parameter list and does not require the creation of a Progress.Lang.ParameterList object at run time.

See also

Assignment (=) statement, Class-based object reference, CLASS statement, DYNAMIC-CAST function, DYNAMIC-INVOKE function, Invoke( ) method (Class), NEW function (classes), New( ) method, NEW statement, NO-ERROR option, Parameter passing syntax, Progress.Lang.ParameterList class

DYNAMIC-NEXT-VALUE function

Returns the next INT64 value of a sequence, incremented by the positive or negative value defined in the specified database.

Syntax

```
DYNAMIC-NEXT-VALUE(
   sequence-expression, logical-dbname-expression [ , tenant-id ] )
```

sequence-expression

A character expression that evaluates to the name of a sequence.

logical-dbname-expression

A character expression that evaluates to the name of a connected database in which the sequence is defined.

tenant-id

An integer expression that evaluates to the tenant ID of a regular tenant, including the default tenant (0). This option applies only to a multi-tenant sequence specified by *sequence-expression* and is intended for access primarily by a super-tenant user.

If you are a regular-tenant user and you specify *tenant-id*, the specified tenant must be the same as the tenant associated with the database connection identity for *logical-dbname-expression* or the AVM raises a run-time error.

If you are a super-tenant user and you do not specify *tenant-id*, the function assumes the effective tenant ID (GET-EFFECTIVE-TENANT-ID function).

If the sequence specified by *sequence-expression* is non-multi-tenant (a shared sequence in a multi-tenant database), and you specify *tenant-id*, the option is ignored.

Notes

- If *logical-dbname-expression* contains the Unknown value (?), the AVM raises a run-time error.
- If *sequence-expression* is a cycling sequence, and the DYNAMIC-NEXT-VALUE function increments the sequence beyond its upper limit (for positive increments) or decrements the sequence beyond its lower limit (for negative increments), the function sets and returns the initial value defined for the sequence.
- If *sequence-expression* is a terminating sequence, and the DYNAMIC-NEXT-VALUE function attempts to increment the sequence beyond its upper limit (for positive increments) or decrement the sequence beyond its lower limit (for negative increments), the function returns the Unknown value (?) and leaves the current sequence value unchanged. Once a sequence terminates, DYNAMIC-NEXT-VALUE continues to return the Unknown value (?) for the specified sequence until it is reset to a new value with the DYNAMIC-CURRENT-VALUE statement, or its definition is changed to a cycling sequence. After changing the sequence definition to cycle, the first use of DYNAMIC-NEXT-VALUE for the sequence sets and returns its initial value.
- If *sequence-expression* is a multi-tenant sequence in the database, each regular tenant has their own current value of the sequence. So, the same values are returned for each tenant that invokes this function.

If the sequence is shared in a multi-tenant database, the values returned by this function are unique across all tenants in the database.

Caution: Be very careful using multi-tenant sequences when you are updating shared tables or multi-tenant tables for a tenant group. For more information, see the sections on using multi-tenant-enabled sequences in *OpenEdge Programming Interfaces*.

- The value of a sequence set by the DYNAMIC-NEXT-VALUE function persists in the database until the next DYNAMIC-CURRENT-VALUE statement or DYNAMIC-NEXT-VALUE function is invoked for the sequence, or until the sequence is deleted from the database.
- You cannot invoke the DYNAMIC-NEXT-VALUE function from within a WHERE clause. Doing so generates a compiler error because the value returned by the DYNAMIC-NEXT-VALUE function can result in ambiguous expressions. To use a result from the DYNAMIC-NEXT-VALUE function in a WHERE clause, assign the result to a variable and use the variable in the WHERE clause instead.
- You can use any combination of the DYNAMIC-NEXT-VALUE function, DYNAMIC-CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, and their non-dynamic versions.

See also

CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, DYNAMIC-CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, NEXT-VALUE function

DYNAMIC-PROPERTY function

Returns the value of a class property. The function allows you to refer to a class property by providing a character expression for the property name, along with an object reference, `THIS-OBJECT` system reference, or the name of a class. The compiler supports both ABL user-defined types and .NET types for the object reference or class name.

At run time, the AVM:

- Determines the actual type of the object reference, if present
- Resolves the property name expression
- Validates the property data type against the type of the target and performs the assignment

Note: The `DYNAMIC-PROPERTY` function also works with .NET indexed properties. Normally, ABL allows you to refer to a .NET indexed property without using the property name, but for dynamic access you must use the property name, typically `Item` indexed property.

Syntax

```
[ return-value = ] DYNAMIC-PROPERTY(
   { object-reference | class-type-name } , property-name [ , index ] )
```

return-value

Specifies a data element that is assigned the value returned when you execute the property's GET accessor. The AVM does the required conversions automatically as if there were a CAST or an ABL conversion function such as STRING or INTEGER present.

object-reference

Specifies a reference to an ABL or .NET class instance that exposes the specified property as an instance member. The compiler allows *object-reference* to be declared as any object type. At run time, the object type must resolve to the type that exposes the property.

class-type-name

Specifies the name of an ABL or .NET class type that defines the specified property as a static member. This is a CHARACTER expression that the AVM evaluates to the type name of a class at run time. It must only be the name of a class: interfaces are not valid.

property-name

Specifies a CHARACTER expression that evaluates to the property name at run time.

Note: The function generates a run-time error if the *property-name* does not name a property exposed by the *object-reference* or a static property exposed by *class-type-name*. A run-time error also gets generated when the required property accessor is not accessible or does not exist. For example, an error is generated when there is an attempt to get the value of a protected property outside the class hierarchy where it is defined, or there is an attempt by the AVM to set a property that has no SET accessor.

index

Specifies an integer expression for the index of the specified element. Use *index* to set or retrieve an individual array element. You can also operate on an entire array by removing the index the same way as you work with non-dynamic property access.

See also

Assignment (=) statement, GetPropertyValue( ) method, SetPropertyValue( ) method

DYNAMIC-PROPERTY statement

Sets the value of a class property. The statement allows you to refer to a class property by providing a character expression for the property name, along with an object reference, THIS-OBJECT system reference, or the name of a class. The compiler supports both ABL user-defined types and .NET types for the object reference or class name.

At run time, the AVM:

- Determines the actual type of the object reference, if present
- Resolves the property name expression
- Validates the property data type against the type of the target and performs the assignment

Note: The DYNAMIC-PROPERTY statement also works with .NET indexed properties. Normally, ABL allows you to refer to a .NET indexed property without using the property name, but for dynamic access you must use the property name, typically Item indexed property.

Note: Compound assignment operators (+=, -=, *=, /=) can be used with the DYNAMIC-PROPERTY statement. For more information, see += Addition assignment operator, -= Subtraction assignment operator, *= Multiplication assignment operator, and /= Division assignment operator.

Syntax

```
DYNAMIC-PROPERTY( { object-reference | class-type-name }
  , property-name [ , index ] ) = new-value
```

object-reference

Specifies a reference to an ABL or .NET class instance that exposes the specified property as an instance member. The compiler allows *object-reference* to be declared as any object type. At run time, the object type must resolve to the type that exposes the property.

class-type-name

Specifies the name of an ABL or .NET class type that defines the specified property as a static member. This is a CHARACTER expression that the AVM evaluates to the type name of a class at run time. It must only be the name of a class: interfaces are not valid.

property-name

Specifies a CHARACTER expression that evaluates to the property name at run time.

index

Specifies an integer expression for the index of the specified element. Use *index* to set or retrieve an individual array element. You can also operate on an entire array by removing the index the same way as you work with non-dynamic property access.

new-value

The value of any data type that can be assigned to the property.

Note: The value of *new-value* cannot be converted to the property's type.

Examples

The following example code uses the DYNAMIC-PROPERTY statement to set the Amount property of the invoice object:

```
VAR InvoiceClass invoice.

invoice = NEW InvoiceClass().

/* Set the Amount property */
DYNAMIC-PROPERTY(invoice, "Amount") = 654.32.
```

The following example code uses the += assignment operator, with the DYNAMIC-PROPERTY statement, to add to the Amount property:

```
VAR InvoiceClass invoice.
VAR DECIMAL latefee = 30.

invoice = NEW InvoiceClass().

/* Add to the Amount property */
DYNAMIC-PROPERTY(invoice, "Amount") += latefee.
```

See also

Assignment (=) statement, GetPropertyValue( ) method, SetPropertyValue( ) method

EDITING phrase

Identifies the process that follows each keystroke during a PROMPT-FOR, SET, or UPDATE statement.

This phrase is maintained primarily for compatibility with Progress Version 6 or earlier.

Note: Does not apply to SpeedScript programming.

Syntax

```
[ label: ] EDITING: statement... END
```

statement

One or more statements you want to process, usually for each keystroke entered. In most cases, the first statement is READKEY.

Example

This procedure lets you update the ix variable, and immediately processes each of your keystrokes. The READKEY statement reads each of the keys you press. The APPLY statement applies, or executes, each keystroke. This is a very simple EDITING phrase and is the same as entering UPDATE ix.

r-edit.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

UPDATE ix EDITING:
  READKEY.
  APPLY LASTKEY.
END.
```

The following r-edit2.p procedure uses an EDITING phrase with an UPDATE statement to control what happens based on each keystroke during the UPDATE. Here, the user can press any key while updating any field except SalesRep.

While in the SalesRep field, the user can press **SPACEBAR** to scroll through the possible values for the SalesRep field. If the user presses the **TAB**, **BACKTAB**, **GO**, **RETURN**, or **END-ERROR** key, the procedure executes that key. If the user presses any other key while in the SalesRep field, the terminal beeps.

r-edit2.p

```
PROMPT-FOR Customer.CustNum.
FIND Customer USING Customer.CustNum.

/* Update Customer fields, monitoring each keystroke during the UPDATE */
UPDATE Customer.Name Customer.Address Customer.City Customer.State SKIP
  Customer.SalesRep HELP "Use the space bar to select a SalesRep"
  WITH 2 COLUMNS EDITING:  /* Read a keystroke */
  READKEY.
  /* If the cursor is in any field except SalesRep, execute the last key
     pressed and go on to the next iteration of this EDITING phrase to check
     the next key */
  IF FRAME-FIELD <> "SalesRep" THEN DO:
    APPLY LASTKEY.
    IF GO-PENDING THEN LEAVE.
    ELSE NEXT.
  END.
  /* When in the SalesRep field, if the last key pressed was the space bar
     then cycle through the sales reps */
  IF LASTKEY = KEYCODE(" ") THEN DO:
    FIND NEXT SalesRep NO-ERROR.
    IF NOT AVAILABLE SalesRep THEN FIND FIRST SalesRep.
    DISPLAY SalesRep.SalesRep @ Customer.SalesRep.
    NEXT.
  END.
  /* If the user presses any one of a set of keys while in the SalesRep field,
     immediately execute that key */
  IF LOOKUP(KEYFUNCTION(LASTKEY),
    "TAB,BACK-TAB,GO,RETURN,END-ERROR") > 0 THEN APPLY LASTKEY.
  ELSE BELL.
END.
```

Notes

- A READKEY statement does not have to be the first statement after the word EDITING. However, it should appear somewhere in the EDITING phrase because the AVM does not automatically read keystrokes when you use an EDITING phrase.
- The EDITING phrase applies to the PROMPT-FOR part of a SET or UPDATE statement. Therefore, to examine a value supplied by the user (within an EDITING phrase), you must use the INPUT function to refer to the field or variable that contains the value.
- When you use the NEXT statement in an EDITING phrase, the AVM executes the next iteration of that EDITING phrase and cancels any pending GO.
- When you use the LEAVE statement in an EDITING phrase, the AVM leaves the EDITING phrase and executes the assignment part of the SET or UPDATE statement.
- Within an EDITING phrase, you cannot use the CLEAR ALL, DOWN, or UP statements on the frame being edited.
- If you hide and redisplay a frame while you are in an EDITING block, the AVM might not redisplay it in the same location unless you specifically name the row and column of the frame. This could cause problems because the EDITING block does not recognize the new location, and attempts to update the fields at the old frame location.
- The EDITING phrase activates only for input from a terminal. If your input comes from an operating system file (set with the INPUT FROM statement), the EDITING phrase has no effect.

- The EDITING phrase is incompatible with event-driven programming. An EDITING block might interfere with other event handling statements.
- For more information on EDITING blocks and other ways of monitoring keystrokes, see *OpenEdge Programming Interfaces*.

See also

END statement, PROMPT-FOR statement, READKEY statement, SET statement, UPDATE statement

---

EDITOR phrase

Specifies that a field or variable is displayed as a text editor widget. This is especially useful for long text (CHARACTER and LONGCHAR) fields. The EDITOR phrase is an option of the VIEW-AS phrase.

Syntax

```
EDITOR
  { size-phrase
    | INNER-CHARS characters INNER-LINES lines
  }
  [ BUFFER-CHARS chars ]
  [ BUFFER-LINES lines ]
  [ LARGE ]
  [ MAX-CHARS characters ]
  [ NO-BOX ]
  [ NO-WORD-WRAP ]
  [ SCROLLBAR-HORIZONTAL ]
  [ SCROLLBAR-VERTICAL ]
  [ TOOLTIP tooltip ]
```

size-phrase

Specifies the outer width and height of the text editor widget in characters or pixels. This is the syntax for size-phrase:

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

For more information, see the SIZE phrase reference entry.

INNER-CHARS *chars* INNER-LINES *lines*

Specifies the number of characters visible in each line of the Editor and the number of lines visible within the Editor. Both *chars* and *lines* must be integer constants.

Note that the values you supply for INNER-CHARS and INNER-LINES specify only the size of the editing area, not the overall size of the editor widget. The overall size is determined by the size of the editing area plus the sizes of the margin and border heights and widths.

BUFFER-CHARS *chars*

In character mode, specifies the number of characters a user can enter on each line. When the last character is typed, the text input cursor automatically wraps to the next line. This option is ignored in graphical environments.

The *chars* value must be an integer constant that is equal to or greater than the value specified by SIZE *width* or INNER-CHARS *chars*. If greater, horizontal scrolling is enabled. The default is the value specified by SIZE *width* or INNER-CHARS *chars*.

BUFFER-LINES *lines*

In character mode, specifies the number of lines a user can enter. By default, ABL does not limit the number of lines (although system limits might apply). This option is ignored in graphical environments.

The *lines* value must be an integer constant that is equal to or greater than the value specified by BY *height* or INNER-LINES *lines*. If equal, vertical scrolling is disabled.

LARGE

Specifies that ABL use a large editor widget rather than a normal editor widget in Windows. A normal Windows editor can contain up to 20K of data. The LARGE option allows the editor to contain data up to the limit of your system resources. However, it also consumes more internal resources and lacks some functionality. Use the LARGE option only if you have to edit very large sections of text. The LARGE option applies only to Windows; other interfaces allow for larger editors by default. This option is ignored in those other interfaces.

MAX-CHARS *characters*

The maximum number of characters that can be displayed or entered within the text editor widget. The value *characters* must be an integer constant. By default, ABL does not limit the number of characters (although system limits might apply).

NO-BOX

Specifies that the editor be displayed without a border. The default is to display the editor with a border. The NO-BOX option has no effect on the size of the editor.

NO-WORD-WRAP

Specifies that word wrap be disabled within the text editor widget. If you enable word wrap, horizontal scrolling is disabled. This option is ignored in character mode. This is the default with the LARGE option.

SCROLLBAR-HORIZONTAL

Specifies that horizontal scrolling is enabled and a horizontal scroll bar is displayed for the widget.

SCROLLBAR-VERTICAL

Specifies that a vertical scroll bar is display for the widget. Although vertical scrolling is always enabled within a text editor widget, a vertical scroll bar is displayed only if you specify this option.

TOOLTIP *tooltip*

Allows you to define a help text message for a text field or text variable. The AVM automatically displays this text when the user pauses the mouse button over a text field or text variable for which a tooltip is defined.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the tooltip is removed. No tooltip is the default. The TOOLTIP option is supported in Windows only.

Example

The following example uses two editor widgets. The `Item.CatDescription` field is viewed as an EDITOR in the item-info frame and the variable `my_clipbd` is viewed as an EDITOR in the clip frame. Use the EDITOR functions provided by your interface environment to copy text from `CatDescription` into my_clipbd. You can then subsequently copy that text into the `CatDescription` field of another item.

r-vaedit.p

```
DEFINE VARIABLE my_clipbd AS CHARACTER NO-UNDO
  VIEW-AS EDITOR SIZE 60 BY 6 SCROLLBAR-VERTICAL LABEL "Scratch Pad".

DEFINE BUTTON b_quit LABEL "Quit" AUTO-ENDKEY.

FORM
  Item.ItemNum
  Item.ItemName
  Item.Price
  Item.OnHand
  Item.Allocated
  Item.ReOrder
  Item.OnOrder
  Item.CatPage
  Item.CatDescription VIEW-AS EDITOR SIZE 35 BY 3 SCROLLBAR-VERTICAL
  WITH FRAME item-info 1 DOWN ROW 1 CENTERED SIDE-LABELS
  TITLE "Update Item Category Description".

FORM my_clipbd WITH FRAME clip.

DEFINE FRAME butt-frame b_quit
  WITH CENTERED ROW 18.

ON GO OF Item.ItemNum DO:
  FIND Item USING Item.ItemNum EXCLUSIVE-LOCK.
  DISPLAY Item WITH FRAME item-info.
  ENABLE Item.CatDescription WITH FRAME item-info.
  ENABLE my_clipbd WITH FRAME clip.
END.

ON GO OF Item.CatDescription DO:
  ASSIGN Item.CatDescription.
  CLEAR FRAME item-info.
  DISABLE Item.CatDescription WITH FRAME item-info.
END.

ENABLE Item.ItemNum WITH FRAME item-info.
ENABLE b_quit WITH FRAME butt-frame.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- If you specify the SCROLLBAR-VERTICAL option in, a vertical scroll bar appears on the side of the Editor. The user can then use the scroll bar to scroll within the widget. Whether or not you specify

SCROLLBAR-VERTICAL, the user can scroll vertically by using the up and down arrow keys to move above or below the displayed text.

- If you use the SIZE phrase to specify the dimensions of the Editor, the AVM uses a portion of this overall space (thereby shrinking the size of the editing area) for any scroll bars you specify. Use the INNER-CHARS and INNER-LINES options if you want a fixed size for the editing area, regardless of the presence of scroll bars.
- In Windows, the editor widget supports lines of up to 255 characters only.
- By default, the editor widget supports text wrap. This means that when you reach the end of a line within the widget, text wraps to the next line rather than scrolling to the right. In graphical interfaces, you can enable horizontal scrolling by specifying either the NO-WORD-WRAP or SCROLLBAR-HORIZONTAL options. If you specify SCROLLBAR-HORIZONTAL, a horizontal scroll bar appears. If you specify NO-WORD-WRAP, but not SCROLLBAR-HORIZONTAL, the user can scroll horizontally by using the left and right arrow keys at the edge of the displayed text.
- Windows allows a user to transfer focus to the editor by pressing **ALT** and one of the letters in the label. This is called a *mnemonic*.
- The character-mode editor does not support the tab character. When the AVM reads a file that contains tabs into an editor widget, it replaces the tabs with eight spaces. When it writes the file, the tabs are not restored and the file is permanently changed.
- When you specify the LARGE option, the following attributes and methods no longer apply to the editor:
  - CONVERT-TO-OFFSET method
  - CURSOR-OFFSET attribute
  - LENGTH attribute
  - MAX-CHARS attribute
  - SELECTION-END attribute
  - SELECTION-START attribute
  - SET-SELECTION method
  - WORD-WRAP attribute
- For SpeedScript, the only valid options are: *size-phrase*, INNER-CHARS, INNER-LINES, MAX-CHARS, NO-BOX, NO-WORD-WRAP.

See also

SIZE phrase, VIEW-AS phrase

EMPTY TEMP-TABLE statement

Empties a temp-table.

When you empty a temp-table that is defined as UNDO within a transaction, the AVM deletes the records individually. This is less efficient than emptying the temp-table outside the transaction, where the AVM deletes all records in the temp-table as a unit.

Note: This statement corresponds to the EMPTY-TEMP-TABLE( ) method.

Syntax

```
EMPTY TEMP-TABLE temp-table-name [ NO-ERROR ]
```

temp-table-name

The name of the temp-table.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

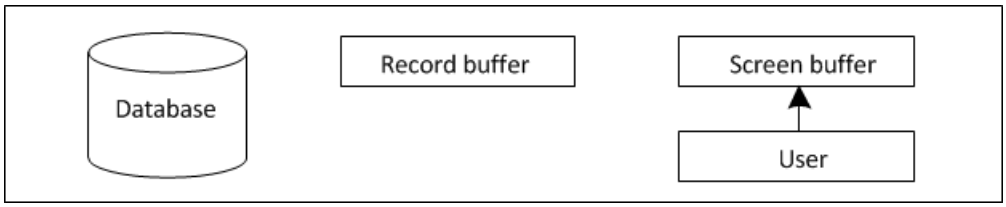
See also

EMPTY-TEMP-TABLE( ) method, NO-ERROR option

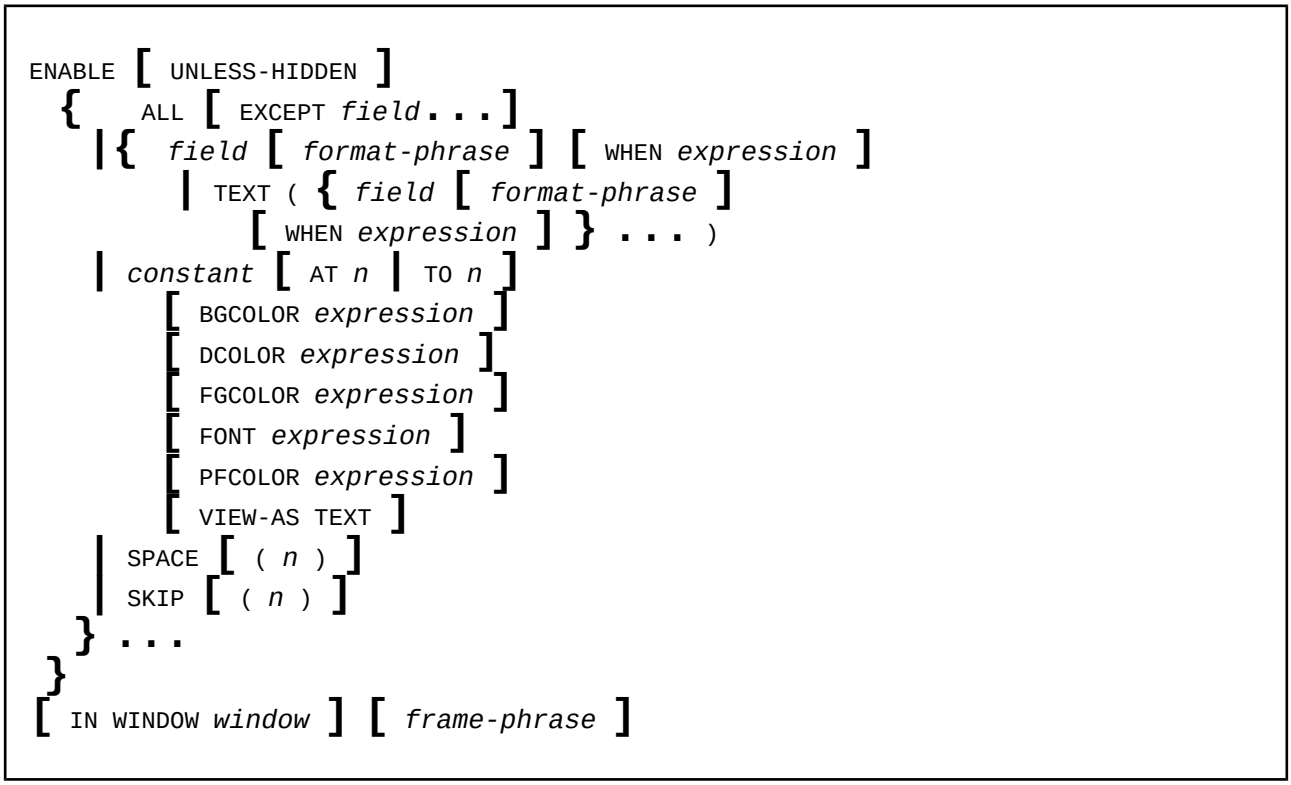
ENABLE statement

Enables input for one or more field-level and child frame widgets within a frame.

Data movement



Syntax



`ALL [ EXCEPT field . . . ]`

Specifies that all field-level widgets for a frame are enabled, except those you list.

`UNLESS-HIDDEN`

Restricts ENABLE to fields whose HIDDEN attribute is FALSE.

`field`

Specifies the name of the field, variable, or widget you want to enable. Remember that the ENABLE statement accepts input only and stores it in the screen buffer. The underlying record buffer of a field or variable is unaffected unless you ASSIGN the value.

In array fields, array elements with constant subscripts are treated just like any other field. Array fields with no subscripts or array fields in the FORM statement are expanded as though you had entered the implicit elements. See the DISPLAY statement reference entry for information on how array fields with expressions as subscripts are handled.

Note: You cannot enable indeterminate array variables with no size.

`format-phrase`

Specifies one or more frame attributes for a field, variable, or expression. For more information on *`format-phrase`*, see the Format phrase reference entry.

WHEN *expression*

Enables the field only if *expression* has a value of TRUE when the ENABLE statement is executed. Here, *expression* is a field name, variable name, or expression that evaluates to a LOGICAL value.

TEXT

Defines a group of character fields or variables (including array elements) to use automatic text-wrap. The TEXT option works only with character fields. When you insert data in the middle of a TEXT field, the AVM wraps data that follows into the next TEXT field, if necessary. If you delete data from the middle of a TEXT field, the AVM wraps data that follows into the empty area.

If you enter more characters than the format for the field allows, the AVM discards the extra characters. The character fields must have formats in the form x(n). A blank in the first column of a line marks the beginning of a paragraph. Lines within a paragraph are treated as a group and will not wrap into other paragraphs.

constant [AT *n* | TO *n*] [BGCOLOR *expression*] [DCOLOR *expression*] [FGCOLOR *expression*] [FONT *expression*] [PFCOLOR *expression*] [VIEW-AS TEXT]

Specifies a constant (literal) value that you want displayed in the frame. If you use the AT option, *n* is the column in which you want to start the display. If you use the TO option, *n* is the column in which you want to end the display. You can use the BGCOLOR and FGCOLOR options in graphical interfaces to define the foreground and background colors to use when the constant is displayed. Similarly, you can use the DCOLOR and PFCOLOR options in character interfaces to define the prompt and display colors to use when the constant is displayed. The font option, for both character and graphical interfaces, defines the font used. If you use the VIEW-AS TEXT option, the constant is displayed as a text widget rather than a fill-in field.

SPACE [(*n*)]

Identifies the number (*n*) of blank spaces to insert after the field displays. The *n* can be 0. If the number of spaces you specify is more than the spaces left on the current line of the frame, the AVM starts a new line and discards any extra spaces. If you do not use this option or *n*, the AVM inserts one space between items in the frame.

SKIP [(*n*)]

Identifies the number (*n*) of blank lines to insert after the field is displays. The *n* can be 0. If you do not use this option, the AVM does not skip a line between expressions unless the expressions do not fit on one line. If you use the SKIP option, but do not specify *n*, or if *n* is 0, the AVM starts a new line unless it is already at the beginning of a new line.

^

Tells the AVM to ignore an input field when input is being read from a file.

IN WINDOW *window*

Specifies the window in which the widgets are enabled. The *window* parameter must be the name of a currently defined window or an expression that evaluates to the handle for a currently defined window.

frame-phrase

The frame that contains the widgets to enable. If you omit *frame-phrase*, the default frame for the current block is assumed. For more information on *frame-phrase*, see the Frame phrase reference entry.

Example

The following example enables the CustNum field and the Quit button in the main procedure. If you press **GO** in the CustNum field and successfully find a record, the trigger disables the CustNum field and enables the CreditLimit field and the Save and Undo buttons. If you choose Save or Undo, the CHOOSE trigger disables the buttons and enables the CustNum field again. Note that if you choose the Save button, the trigger must execute an ASSIGN statement to set the value in the underlying database field.

r-enable.p

```
DEFINE VARIABLE ok AS LOGICAL NO-UNDO.

DEFINE BUTTON b_quit LABEL "Quit" AUTO-ENDKEY.
DEFINE BUTTON b_save LABEL "Save".
DEFINE BUTTON b_undo LABEL "Undo".

DEFINE FRAME butt-frame
  b_save b_undo b_quit
  WITH CENTERED ROW SCREEN-LINES - 2.

FORM customer
  WITH FRAME cust-info SIDE-LABELS CENTERED
  TITLE "Update Customer Credit Limit".

ON CHOOSE OF b_save, b_undo IN FRAME butt-frame DO:
  DISABLE b_save b_undo WITH FRAME butt-frame.
  DISABLE Customer.CreditLimit WITH FRAME cust-info.
  ENABLE Customer.CustNum WITH FRAME cust-info.
  IF SELF:LABEL = "save" THEN
    ASSIGN FRAME cust-info Customer.CreditLimit.
  CLEAR FRAME cust-info NO-PAUSE.
  APPLY "ENTRY" TO Customer.CustNum IN FRAME cust-info.
END.

ON GO OF Customer.CustNum IN FRAME cust-info DO:
  FIND Customer USING Customer.CustNum EXCLUSIVE NO-ERROR.
  IF AVAILABLE Customer THEN DO:
    DISABLE Customer.CustNum WITH FRAME cust-info.
   ENABLE Customer.CreditLimit WITH FRAME cust-info.
   ENABLE ALL WITH FRAME butt-frame.
    DISPLAY Customer WITH FRAME cust-info.
  END.
  ELSE DO:
    MESSAGE "No customer record exists for customer number"
      INPUT Customer.CustNum ", Please re-enter."
      VIEW-AS ALERT-BOX WARNING BUTTONS OK-CANCEL UPDATE OK.
    IF NOT ok THEN
      APPLY "CHOOSE" TO b_quit IN FRAME butt-frame.
  END.
END.

ENABLE Customer.CustNum WITH FRAME cust-info.
ENABLE b_quit WITH FRAME butt-frame.
WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame
  FOCUS Customer.CustNum IN FRAME cust-info.
```

Notes

- For field representation widgets, the ENABLE statement lets you change the widget's SCREEN-VALUE. If you want to save changes to the field itself, you must subsequently use the ASSIGN statement.

  During data entry, a validation expression defined for the field in the database or in a Format phrase executes only if the widget associated with the field receives input focus. Use the VALIDATE( ) method to execute a validation expression defined for a field regardless of whether it receives input focus or not.

- If you invoke the ENABLE statement for a frame, the AVM brings the frame into view unless the HIDDEN attribute for the frame or one of its ancestor frames or windows is TRUE.
- If you invoke the ENABLE statement for the parent frame of a frame family, the field representation widgets and descendant frames owned by the parent frame are all enabled. However, the field representation widgets of the descendant frames remain disabled and visually insensitive. To enable field representation widgets in the descendant frames and make them sensitive, you must invoke ENABLE statements for each of the descendant frames.
- If you specify the KEEP-TAB-ORDER option for a frame, the ENABLE statement has no affect on the tab order for the frame. Otherwise, the ENABLE statement can affect the tab order of widgets within the frame.
- The tab order for fields specified by the ENABLE statement replaces any conflicting tab order established by previous ENABLE statements or by previous settings of the FIRST-TAB-ITEM, LAST-TAB-ITEM, MOVE-AFTER-TAB-ITEM, or MOVE-BEFORE-TAB-ITEM attributes and methods.
- If you specify the ALL option with the ENABLE statement, the tab order of fields corresponds to the order they are specified in the frame definition. Also, the Data Dictionary field validations and help messages are compiled for all fields in the frame, including view-only fields (for example, text widgets).
- If you specify the ENABLE statement with *field* parameters, the specified fields are moved in the tab order to the end of the order specified for the original frame definition, and the tab order of each *field* corresponds to the order in which it is specified in the statement. The following code enables three widgets (a, b, and c) in frame A with the tab order d, e, f, a, b, and c:

```
DEFINE FRAME A a b c d e f.
ENABLE a b c WITH FRAME A.
```

  Note: Note that widgets d, e, and f are not accessible until their SENSITIVE attributes are set to TRUE.

- If you use more than one ENABLE statement to enable widgets within a frame, each widget is added to the end of the tab order as it is enabled. For example, the following code enables three widgets in a frame:

```
ENABLE a.
ENABLE b.
ENABLE c.
```

  This code sets the tab order as a b c. Rearranging the ENABLE statements changes the tab order.

- For SpeedScript, these options are invalid: BGCOLOR, DCOLOR, FGCOLOR, FONT, IN-WINDOW.

See also

DISABLE statement, WAIT-FOR statement (ABL only)

ENCODE function

Encodes a source character string and returns the encoded character string result.

Syntax

```
ENCODE ( expression )
```

expression

An expression that results in a character string value. If you use a constant, you must enclose it in quotation marks (" ").

Example

This procedure uses the ENCODE function to disguise a password that the user enters, and then displays the encoded password:

r-encode.p

```
DEFINE VARIABLE password      AS CHARACTER NO-UNDO FORMAT "x(16)".
DEFINE VARIABLE id            AS CHARACTER NO-UNDO FORMAT "x(12)".
DEFINE VARIABLE n-coded-p-wrd AS CHARACTER NO-UNDO FORMAT "x(16)".

SET id LABEL "Enter user id" password LABEL
  "Enter password" BLANK WITH CENTERED SIDE-LABELS.

n-coded-p-wrd = ENCODE(password).

DISPLAY n-coded-p-wrd LABEL "Encoded password".
```

Notes

- You can use the ENCODE function to encode a string that contains double-byte characters.
- The ENCODE function performs a one-way encoding operation that you cannot reverse. It is useful for storing scrambled copies of passwords in a database. It is impossible to determine the original password by examining the database. However, a procedure can prompt a user for a password, encode it, and compare the result with the stored, encoded password to determine if the user supplied the correct password.
- In order to ensure reliable results, the original encoding and any subsequent encoded comparisons must run in the same code page. In environments with multiple code pages, Progress Software Corporation strongly recommends that programs use the CODEPAGE-CONVERT function so that occurrences of the ENCODE function related to the same strings run in the same code page.
- The output of the ENCODE function is 16 characters long. Make sure the target field size is at least 16 characters long.

ENCRYPT function

Converts source data into a particular format, and returns a MEMPTR containing the encrypted data (a binary byte stream).

Note: You must use the same cryptographic algorithm, initialization vector, and encryption key values to encrypt and decrypt the same data instance.

Syntax

```
ENCRYPT ( data-to-encrypt [ , encrypt-key [ , iv-value [ , algorithm ]]] )
```

data-to-encrypt

The source data to encrypt. The value may be of type CHARACTER, LONGCHAR, RAW, or MEMPTR.

encrypt-key

An optional RAW expression that evaluates to the name of the encryption key (a binary value) to use in encrypting the specified data. If you specify the Unknown value (?), the current value of the SYMMETRIC-ENCRYPTION-KEY attribute is used. If the value of the SYMMETRIC-ENCRYPTION-KEY attribute is also the Unknown value (?), the AVM generates a run-time error.

You can generate this encryption key, based on the PKCS#5/RFC 2898 standard, by using either the GENERATE-PBE-KEY function or the GENERATE-RANDOM-KEY function.

Note: If you use the GENERATE-RANDOM-KEY function to generate an encryption key, be sure to invoke the function before invoking the ENCRYPT function (not within the ENCRYPT function, which would render the key irretrievable).

The AVM compares the size of the specified encryption key to the key size specified by the cryptographic algorithm. If the key sizes are inconsistent, the AVM generates a run-time error.

You are responsible for generating, storing, and transporting this value.

iv-value

An optional RAW expression that evaluates to an initialization vector value to use with the specified encryption key in the encryption operation. Using an initialization vector value increases the strength of the specified encryption key (that is, it makes the key more unpredictable). If you specify the Unknown value (?), the current value of the SYMMETRIC-ENCRYPTION-IV attribute is used.

algorithm

An optional CHARACTER expression that evaluates to the name of the symmetric cryptographic algorithm to use in encrypting the specified data instance. If you specify the Unknown value (?), the current value of the SYMMETRIC-ENCRYPTION-ALGORITHM attribute is used.

For a list the supported cryptographic algorithms, see the SYMMETRIC-SUPPORT attribute reference entry.

See also

DECRYPT function, SECURITY-POLICY system handle

END statement

Indicates the end of a block started with a CASE, CATCH, CLASS, CONSTRUCTOR, DESTRUCTOR, DO, ENUM, FINALLY, FOR, FUNCTION, INTERFACE, METHOD, PROCEDURE, or REPEAT statement or the end of an EDITING phrase, Trigger phrase, or the implementation of a GET or SET property accessor.

Syntax

```
END [ CASE | CATCH | CLASS | CONSTRUCTOR | DESTRUCTOR | ENUM |
      FINALLY | FUNCTION | GET | INTERFACE | METHOD |
      PROCEDURE | SET | TRIGGERS ]
```

```
[ CASE | CATCH | CLASS | CONSTRUCTOR | DESTRUCTOR | ENUM | FINALLY |
FUNCTION | GET | INTERFACE | METHOD | PROCEDURE | SET | TRIGGERS ]
```

An option that can appear following any END statement that terminates the block for a statement or phrase that is specified with the same name.

Example

This procedure contains two blocks, each ending with the END statement:

r-end.p

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.CustNnum Customer.Name Customer.Phone.
  FOR EACH Order OF Customer NO-LOCK:
    DISPLAY Order WITH 2 COLUMNS.
  END.
END.
```

Notes

- Two blocks, the CATCH block and the FINALLY block, are called *end blocks* because they must appear after the last line of executable code and before the END statement. CATCH blocks must appear before FINALLY blocks. See the entries for these two blocks for information about usage.
- If you do not use any END statements in a procedure, ABL assumes that all blocks end at the end of the procedure.
- If you use any END statements in a procedure, you must use one END statement for every block in the procedure.

See also

CASE statement, CATCH statement, CLASS statement, CONSTRUCTOR statement, DEFINE PROPERTY statement, DESTRUCTOR statement, DO statement, EDITING phrase, ENUM statement,FINALLY statement, FOR statement, FUNCTION statement, INTERFACE statement, METHOD statement, PROCEDURE statement, REPEAT statement, Trigger phrase

ENTERED function

Checks whether a frame field has been modified during the last INSERT, PROMPT-FOR, SET, or UPDATE statement for that field, and returns a TRUE or FALSE result.

Note: Does not apply to SpeedScript programming.

Syntax

```
[ FRAME frame ] field ENTERED
```

[FRAME *frame* **]** *field*

The name of the frame field you are checking. If you omit the FRAME option, the field name must be unambiguous.

Example

This procedure goes through the Customer table and prompts the user for a new `CreditLimit` value. The ENTERED function tests the value the user enters. If the user enters a new value, the procedure displays the old and new `CreditLimit` values. If the user enters the same or no value, the value does not change.

r-enter.p

```
DEFINE VARIABLE new-max NO-UNDO LIKE Customer.CreditLimit.

FOR EACH Customer:
  DISPLAY Customer.CustNum Customer.Name Customer.CreditLimit
    LABEL "Current credit limit"
    WITH FRAME a 1 DOWN ROW 1.
  SET new-max LABEL "New credit limit"
    WITH SIDE-LABELS NO-BOX ROW 10 FRAME b.
  IF new-max ENTERED THEN DO:
    IF new-max <> Customer.CreditLimit THEN DO:
      DISPLAY "Changing Credit Limit of" Customer.Name SKIP
        "from" Customer.CreditLimit "to" new-max
        WITH FRAME c ROW 15 NO-LABELS.
      Customer.CreditLimit = new-max.
      NEXT.
    END.
  END.
  DISPLAY "No Change In Credit Limit" WITH FRAME d ROW 15.
END.
```

Notes

- If you type blanks in a field where data has never been displayed, the ENTERED function returns FALSE, a SET or ASSIGN statement does not update the underlying field or variable. Also, if the AVM has marked a field as entered, and the PROMPT-FOR statement prompts for the field again and you do not enter any data, the AVM no longer considers the field entered.
- If you have changed the field's window value since the last INSERT, PROMPT-FOR, SET, or UPDATE statement on that field, the ENTERED function returns FALSE. For example, if you use the DISPLAY statement to change the value of the field, ENTERED no longer returns TRUE.
- Before referencing a widget with the ENTERED function, you must scope the frame that contains that widget. For example, the following code does not compile:

```
/* This code does not compile. */

DEFINE FRAME x
  myint AS INTEGER
  mychar AS CHARACTER.

ON LEAVE OF mychar
  IF mychar ENTERED THEN
    MESSAGE "Character value changed.".

UPDATE myint mychar WITH FRAME x.
```

  The DEFINE FRAME statement does not scope the frame. Therefore, the reference to the ENTERED function in the trigger cannot be evaluated. To fix the problem, reference the frame in a DISPLAY statement before the ON statement.

See also

NOT ENTERED function

ENTRY function

Returns a character string (CHARACTER or LONGCHAR) entry from a list based on an integer position. The data type of the returned value matches the data type of the list element.

Syntax

```
ENTRY ( element , list [ , character ] )
```

element

An integer value that corresponds to the position of a character string in a list of values. If the value of *element* does not correspond to an entry in the list, the AVM raises the ERROR condition. If the value of *element* is the Unknown value (?), ENTRY returns the Unknown value (?). If *element* is less than or equal to 0, or is larger than the number of elements in *list*, ENTRY returns an error.

list

A list of character strings separated with a character delimiter. The *list* can be a variable of type CHARACTER or LONGCHAR. If the value of *list* is the Unknown value (?), ENTRY returns the Unknown value (?).

character

A delimiter you define for the list. The default is a comma. This allows the ENTRY function to operate on non-comma-separated lists. If you use an alphabetic character, this delimiter is case sensitive.

Examples

This procedure returns the day of the week that corresponds to a date the user enters. The WEEKDAY function evaluates the date and returns, as an integer, the day of the week for that date. The ENTRY function uses that integer to indicate a position in a list of the days of the week.

r-entry.p

```
DEFINE VARIABLE datein AS DATE      NO-UNDO.
DEFINE VARIABLE daynum AS INTEGER   NO-UNDO.
DEFINE VARIABLE daynam AS CHARACTER NO-UNDO INITIAL "Sunday,
  Monday, Tuesday, Wednesday, Thursday, Friday, Saturday".

SET datein LABEL "Enter a date (mm/dd/yy)".
daynum = WEEKDAY(datein).
DISPLAY ENTRY(daynum,daynam) FORMAT "x(9)" LABEL "is a" WITH SIDE-LABELS.
```

This is an example of a list separated by dashes instead of commas (the result is "helvetica"):

r-entry2.p

```
DEFINE VARIABLE typeface AS CHARACTER NO-UNDO.

typeface = "-adobe-helvetica-bold-r-normal--*-210-*-*-*-*-iso*-*".
DISPLAY ENTRY(3, typeface, "-") FORMAT "x(16)".
```

The next procedure looks up UNIX login IDs in a small password array and returns the name of the user:

r-entry3.p

```
DEFINE VARIABLE login-name AS CHARACTER NO-UNDO FORMAT "x(10)".
DEFINE VARIABLE real-name  AS CHARACTER NO-UNDO FORMAT "x(20)".
DEFINE VARIABLE loop       AS INTEGER   NO-UNDO.
/* username:password:uid:gid:gcos-field:home-dir:login-shell */
DEFINE VARIABLE passwd     AS CHARACTER NO-UNDO EXTENT 5 INITIAL
 ["kulig::201:120:Clyde Kulig:/users/kulig",
  "gegetskas::202:120:Neal Gegetskas:/users/geget:",
  "bertrand::203:120:Rich Bertrand:/users/bertr:",
  "lepage::204:120:Gary Lepage:/users/lepag:",
  "wnek::205:120:Jordyn Wnek:/users/wnekj:"].

REPEAT:
  SET login-name.
  real-name = ?.
  DO loop = 1 TO 5:
    IF ENTRY(1,passwd[loop],":") = login-name THEN LEAVE.
  END.
  IF loop > 5 THEN
    MESSAGE "Sorry, but" login-name "is not in my password file.".
  ELSE
    real-name = ENTRY(5,passwd[loop],":").
  DISPLAY real-name.
END.
```

Notes

The ENTRY function is double-byte enabled. It can return an entry that contains double-byte characters from a specified list and the *character* delimiter can be a double-byte character.

See also

ENTRY statement, LOOKUP function, NUM-ENTRIES function

ENTRY statement

Used on the left-hand side of an assignment to set the *n*th element to some value.

Syntax

```
ENTRY ( element , list [ , character ] ) = expression
```

element

An integer value that corresponds to the position of a character string in a list of values. If the value of *element* does not correspond to an entry in the list, the AVM raises the ERROR condition. If the value of *element* is the Unknown value (?), ENTRY returns the Unknown value (?). If *element* is less than or equal to 0, or is larger than the number of elements in *list*, ENTRY returns an error.

list

A list of character strings separated with a character delimiter. The *list* can be a variable of type CHARACTER or LONGCHAR. If the value of *list* is the Unknown value (?), ENTRY returns the Unknown value (?).

character

A delimiter you define for the list. The default is a comma. This allows functions to operate on non-comma-separated lists. The delimiter must be only a single character. If you specify a string of more than one character, only the first character is used. If you specify a null string (""), a space character is used as the delimiter. If you use an alphabetic character, this delimiter is case sensitive.

expression

A constant, field name, variable name, or expression that results in a character string whose value you want to store in the *n*th element in a list. ABL does not pad or truncate *expression*.

Note: The += Addition assignment operator can be used with the ENTRY statement.

Example

This procedure uses three ENTRY statements:

r-ent-eq.p

```
DEFINE VARIABLE num-recs AS INTEGER   NO-UNDO.
DEFINE VARIABLE msg-txt  AS CHARACTER NO-UNDO INITIAL
  "There are <x> records in the table.".

/* Count the records. */
FOR EACH Customer NO-LOCK:
  num-recs = num-recs + 1.
END.

/* If there is only one record, make the message singular. */
IF num-recs = 1 THEN
  ASSIGN
    ENTRY(2,msg-txt," ") = "is"
    ENTRY(4,msg-txt," ") = "record".

/* Insert the record count into the string. */
ENTRY(3,msg-txt," ") = STRING(num-recs).

MESSAGE msg-txt.
```

The following example code uses the addition assignment operator (+=) with the ENTRY statement.

```
VAR CHAR myList = "apples,oranges,blueberries".

ENTRY(2,myList) = "peaches".

MESSAGE myList.   //apples,peaches,blueberries

ENTRY(2,myList) += ",kiwi".

MESSAGE myList.   //apples,peaches,kiwi,blueberries
```

Notes

- The ENTRY statement is double-byte enabled. It can insert an entry that contains double-byte characters into a specified list and the *character* delimiter can be a double-byte character.

See also

ENTRY function, LOOKUP function, NUM-ENTRIES function

ENUM statement

Defines a user-defined enumeration (also known as an enumerated type or enum). An enum is made up of a list of strongly typed, named constants called members. The value of a variable defined as an enum is restricted to the list of members defined for that enum. Some of the advantages of enums are that they are self-documenting, strongly typed, and validated at compile time (i.e., the compiler verifies that a variable defined as an enum can only contain values defined by that enum).

In addition to being named, each member also has an underlying numeric value associated with it. By default, the compiler sets those associated values automatically. Generally, you do not need to worry about the numerical values, but you can specify one or more of the values explicitly if desired.

You can define two types of enum. In the standard version, a variable defined as an enum can only be assigned one value at a time. You can also define a flag enum. Any variable defined as a flag enum can be assigned any combination of the members (also referred to as "flags" for flag enums) at one time. You can perform operations on flag enum variables to set and unset one or more of the flags.

All enums implicitly inherit from one of two classes. The single-value enums inherit from `Progress.Lang.Enum`, and flag enums inherit from `Progress.Lang.FlagsEnum`, which is a subclass of `Progress.Lang.Enum`. See the Progress.Lang.Enum class and Progress.Lang.FlagsEnum class entries for more information.

Syntax

```
ENUM enum-type-name [FLAGS]:
    DEFINE ENUM enum-member-definition
               [enum-member-definition] ...
END [ENUM].
```

enum-type-name

Defines the type name for a user-defined enum. You can specify an ABL enum name in a way similar to how you specify an ABL class type. All the requirements and restrictions on how to specify an ABL class type apply to an enum. See the CLASS statement and Type-name syntax entries for more details.

FLAGS

Defines the enum as a flag enum, with the defined members acting as flags. This allows you to set one or more flags into the same variable. You can define up to 64 distinct members to be used as flags. You can also define a flag as a combination of other flags, and a flag defined this way does not count as a distinct flag.

enum-member-definition

Defines a member using the following syntax:

```
member-name [ = {member-name [, member-name]... }
            | constant ]
```

member-name

Specifies the name of the member. Each member name must be unique among the names of all members defined in the enum as well as all properties, events, and variable data members that are defined in the class hierarchy and accessible to the defining class. The rules for member names follow the same rules that are in place for variable and property names. Member names are case insensitive and can be reserved keywords.

member-name [, *member-name*]

Assigns the same numeric value as a previously defined member. In this way, you can define multiple members as equivalent with the same underlying value. For flag enums, you can also equate an member with multiple, previously defined members, entered as a comma-delimited list. See the Examples section for more information.

constant

Defines the numeric value associated with the member. The value must be a constant of type INT64. The value can be represented in hexadecimal notation (for easier readability with flag enums, for example).

If the first member is not explicitly assigned a value, the compiler starts with a value of 1. For each subsequent member not assigned a value, the compiler assigns a value 1 larger than the previous member for a single-value enum, or the next power of 2 for a flag enum.

Note: Flag enums are the primary use case for explicitly setting the underlying values of members. Progress recommends that, when defining a flag enum, you always explicitly define a member with a value of 0.

END [ENUM]

Specifies the end of the enum definition. You must end the enum definition with the END statement.

Examples

The following example shows how to define an enum with the four cardinal directions as its named values:

```
ENUM Direction:
    DEFINE ENUM North
                South
                East
                West.
END ENUM.
```

The next example shows how to define an member as equivalent to another:

```
ENUM Direction:
    DEFINE ENUM North
                Default = North
                South
                East
                West.
END ENUM.
```

In this case, both `North` and `Default` have an underlying value of 1.

The following is an example of a flag enum, including the member `ReadWrite` being defined as a combination of previously defined members:

```
ENUM Permission FLAGS:
    DEFINE ENUM None = 0
                Read
                Write
                Update = Write
                ReadWrite = Read,Write
                Create
                Delete
                Execute.
END ENUM.
```

For flag enums, it is recommended that you include an member with an underlying value of 0, representing no flags set. Because the compiler automatically starts with a value of 1 for both single-value and flag enums, you must define an member with a value of 0 explicitly.

The syntax for assigning a value to a variable defined as an enum is similar to the syntax used to assign values to static data members of a class:

```
enum-type-name:member-name
```

Using the `Direction` enum from the first example, this code fragment defines `myDirection` as a variable of type `Direction` and assigns `myDirection` the member `South`:

```
DEFINE VARIABLE myDirection AS Direction.

myDirection = Direction:South.
```

This syntax cannot be used to set multiple flags for a variable defined as a flag enum. To set multiple flags, you need to use bitwise operators or the `SetFlag( )`, `ToggleFlag( )`, and `UnsetFlag( )` methods, as described in the Notes section of this entry.

Members are read-only, so the following code will not compile:

```
Direction:South = myDirection.
```

Like other uninitialized class references, variables defined as enums have an initial value of the unknown (?) value.

Notes

- Each member represents an instance of an object of the type of enum that defines them. When you reference a member, you get an instance that is maintained by the AVM.
- Because enums are implicitly class based, you can use them in many of the same ways you can use a class type or instance. You can define variables, parameters, return types, and class-based properties as an enum. These data elements can then hold a reference to an enum instance. For example, using the `Permission` enum defined in the Examples, the following code fragments are all valid:

```
METHOD PUBLIC Permission SomeOperation (INPUT perm AS Permission):
END.

PROCEDURE proc:
 DEFINE INPUT-OUTPUT PARAMETER perm AS Permission.
END.

FUNCTION func1 RETURNS Permission (OUTPUT perm AS Permission):
END.
```

  You can explicitly pass members as INPUT parameters using the *enum-type-name*`:`*member-name* syntax, but because the member itself cannot be modified, it cannot be passed as an OUTPUT or INPUT-OUTPUT parameter. For example, the following is not allowed:

```
RUN proc1 (INPUT-OUTPUT Permission:Read).
```

  You can also assign an enum object reference to a temp-table field defined as the Progress.Lang.Object class type, but you cannot assign an enum object reference to a database table field. For more information on object references in general, see the reference entry for a Class-based object reference.

- Enums must be defined in a .cls file by themselves, not with other class or interface definitions.
- Although enums are implicitly class based, you can not instantiate an enum using NEW. You must use the type-name syntax shown in the Examples section to access the members defined in an enum, or use the `GetEnum( )` method to get an enum instance. The method takes an enum member name as a CHARACTER or the underlying numeric value of a member as an INT64 and returns an enum instance. For example, both of the following lines of code create an enum instance using the `Direction` enum type, and in both cases, `myDirection` is assigned the member `South`:

```
DEFINE VARIABLE myDirection AS Direction.

myDirection = Direction:GetEnum("South").
myDirection = Direction:GetEnum(2).
```

  This method is not part of the implementation of `Progress.Lang.Enum`, but it is automatically available for all built-in and user-defined enum types. For full details, see the GetEnum( ) method entry.

- All enums are final. They also cannot explicitly inherit from another class or implement interfaces.
- Enums are serializable. You can pass parameters defined as enums between an ABL client and an application server.
- Much of the functionality for classes is also supported for enums, including package names, the USING statement, reflection, and assignment and casting rules. Unlike classes, however, enums cannot have data members, properties, events, or user-defined methods.

- You can define variables, properties, and parameters using the AS CLASS phrase. For example:

```
DEFINE VARIABLE myDirection AS CLASS Direction.
```

- In the original `Direction` enum example, the underlying values for the members are automatically set by the compiler as 1 for `North`, 2 for `South`, and so on. However, you can change this pattern by explicitly setting one or more of the numerical values. For example:

```
ENUM Direction:
    DEFINE ENUM North
                South
                East = 4
                West.
END ENUM.
```

  In this case, the underlying values for the members are still automatically set by the compiler as 1 for `North`, 2 for `South`. `East` is explicitly set as 4, so the compiler sets the value of `West` as one larger than the previous member, which in this case will be 5. The same rules hold true for flag enums, except that the compiler assigns the next power of 2 to each member that does not have its underlying value explicitly set.

- When you have to explicitly set the underlying value of an member, you can use hexadecimal notation, particularly for ease of readability with flag enums. Following is the `Permission` enum with the underlying values explicitly set:

```
ENUM Permission FLAGS:
    DEFINE ENUM None = 0
                Read = 0x01
                Write = 0x02
                Update = 0x02
                ReadWrite = 0x03
                Create = 0x04
                Delete = 0x08
                Execute = 0x10.
END ENUM.
```

- If desired, you can split the defined list of members in an ENUM statement into two or more DEFINE ENUM statements for readability. For example:

```
ENUM Direction:
    DEFINE ENUM North
                South.
    DEFINE ENUM East
                West.
END ENUM.
```

  The same rules for how the compiler assigns the underlying numerical values still apply, i.e., in the example above, `South` has an underlying value of 2, and `East` has an underlying value of 3. In other words, the underlying numerical values will be assigned according to the order of the members, regardless of whether or not the member list is split into multiple DEFINE ENUM statements.

Note: Although you can split the list of members into separate DEFINE ENUM statements, the list of members for a given enum must be completely defined in a single ENUM statement. Additional members cannot be added to a defined enum in a separate ENUM statement.

- The following comparison operators can be used with enums, as long as the two expressions being compared are both defined as the same enum type:
  - EQ (=)
  - NE (<>)
  - GT (>)
  - LT (<)
  - GE (>=)
  - LE (<=)

  When used with expressions that are enum types, the operators compare the underlying numerical values of the members. Comparisons can be made using either single-value enums or flag enums. Using the previously defined `Direction` enum, this code fragment shows a comparison and results in the message being displayed:

```
DEFINE VARIABLE myDirection AS Direction.

myDirection = Direction:South.

IF myDirection NE Direction:North THEN
   MESSAGE "The current heading is not north."
      VIEW-AS ALERT-BOX.
```

  You can only use the comparison operators for two expressions defined as the same enum type.

  The CompareTo( ) method of `Progress.Lang.Enum` also allows you to compare enum instances based on the members' numeric values. See the CompareTo( ) method entry for details.

- The CASE statement also supports the use of enums. For example, using the `Direction` enum defined in the Examples section:

```
DEFINE VARIABLE myDirection AS Direction.

myDirection = Direction:South.

CASE myDirection:
   WHEN Direction:North THEN
      MESSAGE "Your current direction is north.".
   WHEN Direction:South THEN
      MESSAGE "Your current direction is south.".
   OTHERWISE
      MESSAGE "Your current direction is east, west, or unknown.".
END CASE.
```

  The expressions in the WHEN phrases must be members from the enum type in the CASE statement.

Note: You can use flag enums with the CASE statement, but be aware that the WHEN phrases check for equality of the members strictly based on numeric values, not as combinations of multiple flags. For example, in the following code, the two WHEN statements for `Permission:Read` and `Permission:Write` will not fire for the given value of `myPermission`; the OTHERWISE statement will.

```
DEFINE VARIABLE myPermission AS Permission.

myPermission = Permission:ReadWrite.

CASE myPermission:
   WHEN Permission:Read THEN
      MESSAGE "You can read data.".
   WHEN Permission:Write THEN
      MESSAGE "You can write data.".
   OTHERWISE
      ...
END CASE.
```

- The following bitwise operators can be used with flag enums. Each operator does a bitwise comparison of two enum expressions and returns a new enum instance as a result:
  - Bitwise AND: Check the state of a particular flag.
  - Bitwise OR: Set a flag.
  - Bitwise XOR (exclusive OR): Toggle a flag.
  - Bitwise NOT (complement): Turn off a flag.

  Caution: Because bitwise NOT turns off flags, it is possible to end up with an enum instance with no flags set, i.e., a value of 0. Always defining an member with value 0 is recommended to avoid an enum expression that doesn't correspond to any defined members.

  As with the logical comparison operators, both expressions must be the same enum type. If they are not, the code will not compile.

Using the Permission enum from the Examples section, the following examples show use cases of the bitwise operators:

```
DEFINE VARIABLE myPermissions AS Permission.

/* myPermissions is set with a Permission instance with the Read and
   Write flags set. */
myPermissions = Permission:Read OR Permission:Write.

/* All flags are set with the possible exception of Create, which is
   toggled. */
myPermissions = myPermissions XOR Permission:Create.

/* Ensures that the Create flag is not set. */
myPermissions = myPermissions AND NOT Permission:Create.

/* All flags are set except Create. */
myPermissions = NOT Permission:Create.

/* Checks to see if Read is set. */
IF (myPermissions AND Permission:Read) = Permission:Read THEN ...

/* Three additional ways to check if Read is set. */
IF (myPermissions AND Permission:Read):Equals(Permission:Read) THEN ...

IF myPermissions:IsFlagSet(Permission:Read) THEN ...

IF (myPermissions AND Permission:Read) NE Permission:None THEN ...
```

The order of operator precedence for bitwise and logical operators is the following:

- bitwise NOT
- logical NOT
- bitwise AND
- bitwise XOR
- bitwise OR
- logical AND
- logical OR

In practice, the order of precedence for bitwise and logical operators are independent because the bitwise operators return an enum instance and the logical operators return a logical. Code like the following will not compile because enum1 AND enum2 returns an enum instance, and a logical is required for the OR part of the statement:

```
IF enum1 AND enum2 OR iCount <> 1...
```

See the Expression entry for the full table of operator precedence.

- In addition to the bitwise operators, there are three methods that you can use to set and unset flags: the ToggleFlag( ) method, the SetFlag( ) method, and the UnsetFlag( ) method. These methods are not part of the implementation of Progress.Lang.FlagsEnum, but they are automatically available for all built-in and user-defined flag enums.
- The MINIMUM and MAXIMUM functions do not support enums.

- The STRING and QUOTER functions return a member name for enum-valued expressions. For flag enum-valued expressions with multiple flags set, STRING returns a comma-separated list of member names. Similarly, PUT will send the appropriate member name(s) to the output destination.
- The INTEGER, INT64, and DECIMAL functions return the underlying numeric value of an member for enum-valued expressions. Given that the numeric values of an member are of type INT64, it is possible the INTEGER function will raise an error with an member if the numeric value cannot be represented as an INTEGER.
- Calling DELETE OBJECT on an enum instance will be silently ignored because enum instances are maintained by the AVM. Enums also do not appear in the SESSION-FIRST-OBJECT/SESSION-LAST-OBJECT chain.

See also

AND operator (bitwise), Assignment (=) statement, Class-based object reference, DYNAMIC-ENUM function, NOT operator (bitwise), , OR operator (bitwise), Progress.Lang.Enum class, Progress.Lang.FlagsEnum class, Type-name syntax, USING statement, XOR operator (bitwise)

Enumeration member access

References an ABL or .NET enumeration member in order to read its value.

Syntax

```
enum-type-name : enumeration-member
```

enum-type-name

The type name of an ABL or .NET enumeration type whose member you want to access. Specify an enumeration type name using the syntax as described in the Type-name syntax reference entry. With an appropriate USING statement, you can specify an unqualified enumeration type name.

enumeration-member

Specifies a name that identifies a specific enumeration member defined in the ABL or .NET enumeration type.

Example

To set the background color of a button (`button1`) to the known system color, aqua, you might enter this ABL code to set the BackColor button property to the value of the appropriate System.Drawing.KnownColor member:

```
button1:BackColor = System.Drawing.KnownColor:Aqua.
```

Similarly, you can set a variable defined as a Progress.Reflect.Flags enumeration to the `Public` member for use with one of ABL's reflection methods:

```
DEFINE myAccessLevel AS Progress.Reflect.Flags.

myAccessLevel = Progress.Reflect.Flags:Public.
```

Notes

- A .NET enumeration type can be an inner member of a class or its own type. For more information on referencing enumeration types, see the Type-name syntax reference entry.
- For a list of ABL constructs and operators that can be used directly with ABL enumeration types, see the ENUM statement entry.

See also

ENUM statement, Progress.Util.EnumHelper class

EQ or = operator

Returns a TRUE value if two expressions are equal.

Syntax

```
expression { EQ | = } expression
```

expression

A constant, field name, variable name, or expression. The expressions on either side of the EQ or = must be of the same data type, although one might be an integer and the other a decimal.

Example

This procedure prompts for the initials of a sales rep. The FOR EACH block reads all the order records for that sales rep. The DISPLAY statement displays information from each of the retrieved records.

r-eq.p

```
PROMPT-FOR Order.SalesRep WITH SIDE-LABELS CENTERED.

FOR EACH Order NO-LOCK WHERE Order.SalesRep EQ INPUT Order.SalesRep:
  DISPLAY Order.OrderNum Order.CustNum Order.OrderDate Order.PromiseDate
    Order.ShipDate
    WITH CENTERED.
END.
```

Notes

- By default, the AVM uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (`-cpcoll`) startup parameter take precedence over a

collation specified for any database the AVM accesses during the session, except when the AVM uses or modifies pre-existing indexes. If you do not specify a collation with the `-cpcoll` startup parameter, the AVM uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, the AVM uses the collation rules with the default name "`basic`" (which might or might not exist in the `convmap.cp` file).

- You can compare character strings with EQ. Most character comparisons are case insensitive in ABL. That is, upper-case and lower-case characters have the same sort value. However, it is possible to define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If either *`expression`* is a field or variable defined as case sensitive, the comparison is case sensitive and "Smith" does not equal "smith".
- Characters are converted to their sort code values for comparison. Using the default case-sensitive collation table, all uppercase letters sort before all lowercase letters (for example, a is greater than Z, but less than b.) Note also that in character code uppercase A is less than [, \ , ^ , _, and ` , but lowercase a is greater than these.
- If one of the expressions has an Unknown value (?) and the other does not, the result is FALSE. If both have the Unknown value (?), the result is TRUE. However, for SQL, if the value of either or both expressions is the Unknown value (?), then the result is the Unknown value (?).
- The equal comparison ignores trailing blanks. Thus, "abc" is equal to "abc ". However, leading and embedded blanks are treated as characters and " abc" is not equal to "abc".
- You cannot compare data of different DATE, DATETIME, and DATETIME-TZ data types to each other using EQ. You must first convert different date and datetime data types to the same data type before doing a comparison between them.
- You can use EQ to compare one BLOB field to another. The AVM performs a byte-by-byte comparison.
- You can use EQ to compare a LONGCHAR variable to another LONGCHAR or CHARACTER variable. The variable values are converted to `-cpinternal` for comparison and must convert without error, or the AVM raises a run-time error.
- You can use EQ to compare a CLOB field to another CLOB field.
- You can use EQ to compare two enums, including both ABL and .NET enums, as long as they are the same enum type. EQ compares the underlying numerical values of the enumerators.

ERROR function

Indicates whether an error occurred during a FILL or SAVE-ROW-CHANGES operation on the specified ProDataSet temp-table buffer.

Syntax

```
ERROR( buffer-name )
```

`buffer-name`

The name of a ProDataSet temp-table buffer.

Notes

- The ERROR function corresponds to the ERROR attribute.

- You can invoke the ERROR function from within a WHERE clause (unlike the corresponding attribute).

ETIME function

Returns, as an INT64 value, the time (in milliseconds) elapsed since the ABL session began or since ETIME (elapsed time) was last set to 0. To set ETIME to 0, pass it a positive logical value, such as YES or TRUE.

Syntax

```
ETIME [ ( logical ) ]
```

logical

A logical value, such as YES or TRUE. The default value is NO.

Examples

This procedure displays the time that elapsed since you began your ABL session:

r-etime.p

```
DISPLAY ETIME.
```

This procedure sets ETIME to 0, runs a procedure called `applhelp.p`, and displays the elapsed time, which, in this case, equals the time required to execute `applhelp.p`:

r-etime2.p

```
DEFINE VARIABLE a AS INT64 NO-UNDO.

DO:
  a = ETIME(yes).
  RUN applhelp.p.
  DISPLAY ETIME.
END.
```

Notes

- ETIME is accurate to at least one-sixtieth of a second, but accuracy varies among systems.
- ABL resets ETIME during startup, not immediately after you enter the pro command. Therefore, the time returned is only an approximation of the time elapsed since your session began.

See also

ADD-INTERVAL function, DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, INTERVAL function, ISO-DATE function, MONTH function, MTIME function, NOW function, TIME function, TIME-SOURCE attribute, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

EXP function

Returns the result of raising a number to a power. The number is called the *base* and the power is called the *exponent*.

Syntax

```
EXP ( base , exponent )
```

base

A constant, field name, variable name, or expression that evaluates to a numeric value.

exponent

A numeric expression.

Example

This procedure calculates how much a principal amount invested at a given compounded annual interest rate grows over a specified number of years:

r-exp.p

```
DEFINE VARIABLE principal AS DECIMAL NO-UNDO FORMAT "->>>,>>9.99"
  LABEL "Amt Invested".
DEFINE VARIABLE rate      AS INTEGER NO-UNDO FORMAT "->9"
  LABEL "Interest %".
DEFINE VARIABLE num-yrs   AS INTEGER NO-UNDO FORMAT ">>9"
  LABEL "Number of Years".
DEFINE VARIABLE final-amt AS DECIMAL NO-UNDO FORMAT
  "->>>,>>>,>>>,>>>,>>>,>>9.99" LABEL "Final Amount".

REPEAT:
  UPDATE principal rate num-yrs.
  final-amt = principal * EXP(1 + rate / 100,num-yrs).
  DISPLAY final-amt.
END.
```

Notes

- After converting the base and exponent to the floating-point format, the EXP function uses standard system library routines. On some machines, these routines do not handle large numbers well and might cause your terminal to hang. Also, because the calculations are done in floating-point arithmetic, full decimal precision is not possible beyond 1-12 significant digits on most machines.
- The EXP function is precise to approximately 10 decimal points.

EXPORT statement

Converts data to a standard character format and displays it to the current output destination (except when the current output destination is the screen) or to a named output stream. You can use data exported to a file in standard format as input to other ABL procedures.

Syntax

```
EXPORT [ STREAM stream | STREAM-HANDLE handle ] [ DELIMITER character ]
   {expression...
      | record [ EXCEPT field... ]
    }
  [ NO-LOBS ]
```

```
EXPORT [ STREAM stream | STREAM-HANDLE handle ] { memptr | longchar }
```

`STREAM` *`stream`*

The name of a stream. If you do not name a stream, the AVM uses the unnamed stream. See the DEFINE STREAM statement reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams.

`STREAM-HANDLE` *`handle`*

Specifies the handle to a stream. If *`handle`* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

`DELIMITER` *`character`*

The character to use as a delimiter between field values. The *`character`* parameter must be a quoted **single** character. The default is a space character.

If you specify more than one character as a delimiter, ABL uses the first character as the delimiter.

`expression` `. . .`

One or more expressions that you want to convert into standard character format for display to an output destination.

`record`

The name of the record buffer with fields that you want to convert into the standard character format to display to an output destination.

To use EXPORT with a record in a table name used in multiple databases, you must qualify the record's table name with the database name. See the *`record`* definition in the Record phrase reference entry for more information.

EXCEPT *field* . . .

The AVM exports all fields except those fields listed in the EXCEPT phrase.

memptr

A variable of data type MEMPTR that contains the data to export.

longchar

A variable of data type LONGCHAR that contains the data to export.

NO-LOBS

Directs the AVM to ignore large object data when exporting records that contain BLOB or CLOB fields.

Examples

This procedure converts the data in the Customer table into standard character format and sends that data to the customer.d file:

r-exprt.p

```
OUTPUT TO customer.d.
FOR EACH Customer NO-LOCK:
  EXPORT Customer.
END.
OUTPUT CLOSE.
```

The next procedure shows how each EXPORT statement creates one line of data (that is, fields are not wrapped onto several lines):

r-exprt2.p

```
OUTPUT TO custdump.
FOR EACH Customer NO-LOCK:
  EXPORT Customer.CustNum Customer.Name Customer.CreditLimit.
END.
OUTPUT CLOSE.
```

That procedure creates a text file, `custdump`, with one line for each Customer. This is a typical line of output:

```
1 "Lift Line Skiing" 58400
```

Use the DELIMITER option to specify a character other than a space to separate fields in the output file. For example, the following procedure uses a semicolon:

r-cstout.p

```
OUTPUT TO custdump2.
FOR EACH Customer NO-LOCK:
  EXPORT DELIMITER ";" Customer.CustNum Customer.Name Customer.CreditLimit.
END.
OUTPUT CLOSE.
```

This is a typical line of output from this code:

```
1;"Lift Line Skiing";58400
```

The following example displays using a MEMPTR to EXPORT mixed character and binary data:

r-expmem.p

```
/* Character and binary data mixed */
DEFINE VARIABLE z AS MEMPTR NO-UNDO.

ASSIGN
  SET-SIZE(z)      = 100
  PUT-STRING(z,1)  = "hi there"
  PUT-LONG(z,10)   = 235
  PUT-STRING(z,14) = "afterint"
  PUT-LONG(z,22)   = 76.

OUTPUT TO abc BINARY NO-CONVERT.
EXPORT z.
OUTPUT CLOSE.
```

Notes

- The EXPORT statement must follow an OUTPUT TO statement, which redirects the output destination.
- Other procedures can use the data exported with the EXPORT statement as input by reading the file with the INSERT, PROMPT-FOR, SET, UPDATE or IMPORT statements, naming one field or variable to correspond to each data element.
- The data is in a **standard format** to be read back into ABL. All character fields are enclosed in quotes ("") and quotes contained in the data you are exporting are replaced by two quotes (""). A single space separates one field from the next. An Unknown value (?) is displayed as an unquoted question mark (?).
- There are no trailing blanks, leading zeros, or formatting characters (for example, dollar signs) in the data.
- ABL exports logical fields as the value YES or NO.
- A Format phrase with an EXPORT statement is ignored.
- If you use a single qualified identifier with the EXPORT statement, the Compiler first interprets the reference as *dbname.tablename*. If the Compiler cannot resolve the reference as *dbname.tablename*, it tries to resolve it as *tablename.fieldname*.
- When exporting fields, you must use table names that are different from field names to avoid ambiguous references. See the Record phrase reference entry for more information.
- When exporting RECID fields, you must explicitly state the RECID field name in the EXPORT statement.
- When exporting ROWID variables or fields in a work table, you must convert the ROWID variable or field to a character string using the STRING function.

- When exporting a LONGCHAR variable, the code page name is written to the header of the export file.
- Never use APPEND while EXPORTing a LONGCHAR variable.
- When exporting records that contain a BLOB or CLOB field, the AVM creates a separate object data file with the code page embedded in the filename and with a file extension of `.blb`. The filename is stored in the BLOB or CLOB field of the exported record. (When importing records that contain a BLOB or CLOB field, the AVM uses this filename to locate the object data file associated with each record.) If the BLOB or CLOB field contains the Unknown value (?), the AVM stores the Unknown value (?) in the BLOB or CLOB field of the exported record, and does not create an object data file. If the BLOB or CLOB field contains a zero-length object, the AVM creates a zero-length object data file.

  The AVM raises the ERROR condition if an object data file cannot be created.
- When dealing with Unicode CLOB and LONGCHAR variables, avoid using the EXPORT statement. The AVM uses the UTF-8 codepage for all Unicode EXPORTs. Use the COPY-LOB statement to avoid this conversion.
- The EXPORT statement creates large object data files in the directory specified as the output destination in the OUTPUT TO statement, by default. You can use the `LOB-DIR` option on the OUTPUT TO statement to specify the directory in which the EXPORT statement creates the BLOB and CLOB data files.
- Use the `NO-LOBS` option with the EXPORT statement to ignore large object data when exporting records that contain BLOB or CLOB fields. When you specify the `NO-LOBS` option, the AVM stores the Unknown value (?) in the BLOB or CLOB field of the exported records and does not create the associated object data files.
- When exporting DATETIME and DATETIME-TZ data, the data format is fixed and conforms to the ISO 8601 standard for date/time representations (`YYYY-MM-DDTHH:MM:SS.SSS+HH:MM`). For DATETIME, there is no time zone offset.
- If you use the DELIMITER option of the EXPORT statement to specify a delimiter other than a space character, you must specify the same delimiter character in a subsequent IMPORT statement that loads the data.
- EXPORT is sensitive to the Date format (-d), Century (-yy), and European numeric (`-E`) startup parameters. When loading data with the IMPORT statement, use the same settings that you used with the EXPORT statement.
- In the MEMPTR version of the EXPORT statement, the MEMPTR's size will determine how much is written to the file. If the size of a MEMPTR is 100, and it only contains a string of length 10, the entire 100 bytes will still be written to the file. The PUT-BYTES statement and GET-BYTES function may be used to move portions of MEMPTRs to areas with varying sizes.You can read and write parts of a file by using MEMPTRs of varying sizes, and multiple EXPORT/IMPORT statements on the same file.

See also

COPY-LOB statement, DEFINE STREAM statement, DISPLAY statement, IMPORT statement, OUTPUT CLOSE statement, OUTPUT TO statement, PUT statement, Stream object handle, STRING function

Expression

A combination of one or more terms and operators that evaluate to a single value of a specific data type.

Syntax

```
[ unary-operator ] [ ( ]{term |term binary-operator term}[ ) ]
```

unary-operator

Any ABL unary operator that can precede the specified *term*, including the:

- + Unary positive operator
- - Unary negative operator
- NOT operator (logical)
- NOT operator (bitwise)

(

Begins a group of one or more terms and operators that are evaluated together before any terms or operators to the left of the '('. If specified, the group must be terminated with a ')'.

term

Any one of the following ABL elements that represents or returns a value with a data type that is compatible with the expression data type, including:

- A literal (constant) value represented according to its data type (Data types)
- A database or temp-table field reference (Record phrase, DEFINE TEMP-TABLE statement)
- A reference to a variable scoped to the current procedure, user-defined function, or method of a class, or to an accessible class-based variable data member, including a subscripted or unsubscripted array reference (VAR statement, DEFINE VARIABLE statement, Class-based data member access)
- A reference to a parameter (of any mode) defined for the current procedure, user-defined function, or method of a class, including a subscripted or unsubscripted array reference (DEFINE PARAMETER statement)
- A reference to a readable class-based property or COM property, including a subscripted or unsubscripted array reference (DEFINE PROPERTY statement, Class-based property access, Accessing COM object properties and methods)
- Readable handle attribute reference (Accessing handle attributes and methods)
- Readable system handle reference (Handle Reference)
- An ABL built-in or user-defined function call (FUNCTION statement)
- A handle method, non-VOID COM method, or non-VOID class-based method call (Accessing handle attributes and methods, Accessing COM object properties and methods, Class-based method call)
- An Expression (recursive)

  Note that each *term* of the expression can itself be formed from an expression, resulting in multiple possible combinations of basic terms and operators, with or without grouping parentheses. Note that each of these elements, except for a more complex Expression, represents a *basic term* of the expression.

binary-operator

Any ABL binary operator that can separate two *term* elements, each of which must have a data type that is compatible with an expression that can result when applying the specified *binary-operator* to the specified *term* elements, including the:

- + Addition operator
- + Concatenation operator
- + Date addition operator
- + Datetime addition operator
- - Subtraction operator
- - Date subtraction operator
- - Datetime subtraction operator
- * Multiplication operator
- / Division operator
- AND operator (bitwise)
- AND operator (logical)
- BEGINS operator
- EQ or = operator
- GE or >= operator
- GT or > operator
- LE or < = operator
- LT or < operator
- MATCHES operator
- MODULO operator
- NE or <> operator
- OR operator (logical)
- OR operator (bitwise)
- XOR operator (bitwise)

)

Terminates a group of one or more terms and operators that are evaluated together before any terms or operators to the right of the ‘)'. If specified, the group must begin with a ‘('.

For information on the precedence of operator evaluation, see Table 37: ABL operator precedence.

Example

The procedure, `r-expression.p`, evaluates an expression to identify the value of an element in an ABL single-dimensional array (`iArray`) that is specified using two-dimensional coordinates (`iDim1` and `iDim2`):

r-expression.p

```
DEFINE VARIABLE iDim1       AS INTEGER   NO-UNDO.
DEFINE VARIABLE iDim2       AS INTEGER   NO-UNDO.
DEFINE VARIABLE iDim1Extent AS INTEGER   NO-UNDO INITIAL 3.
DEFINE VARIABLE iDim2Extent AS INTEGER   NO-UNDO INITIAL 5.
DEFINE VARIABLE iArray      AS CHARACTER NO-UNDO EXTENT 15
  INITIAL ["A","B","C","D","E",  /* iDim1 = 1 */
           "F","G","H","I","J",  /* iDim1 = 2 */
           "K","L","M","N","O"]. /* iDim1 = 3 */

DO iDim1 = 1 TO iDim1Extent:
  DO iDim2 = 1 TO iDim2Extent:
    MESSAGE "iArray[" iDim1 "," iDim2 "]" " = "
      iArray[ iDim2 + (iDim1 - 1) * iDim2Extent ] VIEW-AS ALERT-BOX.
  END.
END.
```

This procedure contains several expressions in the DO statements as well as in the MESSAGE statement. The expression that evaluates the index on the term, `iArray`, using a two-dimensional coordinate is shown in bold. This expression first evaluates the subtraction (`-`) operation in parentheses, then, in order by operator precedence, evaluates the multiplication (*) operation followed by the addition operation (+).

Notes

- A series of terms and operators in an expression, with no grouping parentheses (‘(' and ‘)'), evaluate in an order of precedence determined by the operators. Thus, operators of higher precedence evaluate before operators of lower precedence, and operators of the same precedence evaluate in order from left to right. Otherwise, a series of terms and operators within a particular grouping parentheses evaluate according to their precedence before any series of terms and operators outside that grouping parentheses. The following table shows the order of precedence for ABL operators identified by function and symbol.

Table 37: ABL operator precedence

Precedence (highest to lowest)	Operator function	Symbol
11	Numeric negative (unary)	-
	Numeric positive (unary)	+
10	Numeric modulo	MODULO
	Numeric division	/
	Numeric multiplication	*
9	Date subtraction	-
	Datetime subtraction	-
	Numeric subtraction	-
	Date addition	+
	Datetime addition	+
	Numeric addition	+
	String concatenation	+

Precedence (highest to lowest)	Operator function	Symbol
8	Relational string match Relational less than Relational less than or equal to Relational greater than Relational greater than or equal to Relational equal to Relational not equal to Relational string beginning	MATCHES LT or < LE or <= GT or > GE or >= EQ or = NE or <> BEGINS
7	Bitwise NOT (unary)	NOT
6	Logical NOT (unary)	NOT
5	Bitwise AND	AND
4	Bitwise XOR	XOR
3	Bitwise OR (inclusive)	OR
2	Logical AND	AND
1	Logical OR (inclusive)	OR

Note that before the ABL Virtual Machine (AVM) applies the operators to an expression, it evaluates all the basic terms (variables, properties, methods, functions, etc.) of the expression individually from left to right.

- If you do not include any white space in expressions, ABL does not distinguish between a hyphen (-) used as an operator and a hyphen used as part of an ABL built-in or user-defined element name. In order to enable ABL to identify a hyphen used as an operator, you must add at least one space between the hyphen and the term or terms that the operator evaluates.
- An expression of the appropriate data type can appear as a variable option of any ABL statement that takes an expression, except where specified. Expressions commonly appear in the Assignment (=) statement, ASSIGN statement, and as arguments to INPUT parameters of procedures, functions, and methods (Parameter passing syntax). Expressions cannot appear where a variable syntax option accepts only a quoted string, a user-defined name (such as a procedure or variable name), or a compile-time constant. Other restrictions might also apply.
- You cannot directly reference a BLOB or CLOB database or temp-table field as a term in an expression, except as specifically provided in a given ABL statement or built-in function. Generally, you must first convert the BLOB or CLOB to a MEMPTR or LONGCHAR data type using the COPY-LOB statement, then use the result in the expression.
- When a run-time error occurs in any part of an expression, the AVM stops evaluating the expression and raises the ERROR condition on the statement that contains the expression. If the error occurs in a basic term of the expression, because the AVM first evaluates all the basic terms of an expression from left to right, all such terms (especially, methods and functions) prior to the term that raises ERROR will complete execution. However, any data element set to receive the result of the expression evaluation remains unchanged, for example, when the expression is on the right-hand side of an Assignment (=) statement or

is passed to an INPUT parameter. The ERROR condition from an expression can be handled, like any statement error, as appropriate for the specified statement and block error processing, including use of statement NO-ERROR options, ON ERROR phrases, and CATCH statements to handle any error messages and error objects returned from the expression. For more information, see *ABL Error Handling*.

See also

Assignment (=) statement, ASSIGN statement, CATCH statement, COPY-LOB statement, Data types, ERROR-STATUS system handle, Parameter passing syntax

EXTENT function

This function returns the size (extent) of an array field or variable as an INTEGER value. More specifically, it returns:

- The number of elements for a field or variable defined as a determinate array
- The Unknown value (?) for a variable defined as an indeterminate array with no size
- The size that has been set for a variable defined as an indeterminate array
- Zero for a field or variable that is not an array

Note: The EXTENT function corresponds to the EXTENT attribute.

Syntax

```
EXTENT ( array )
```

array

Any array field or variable.

Example

In the following example, the EXTENT function is used to set the limit of a DO loop that cycles through all elements of an array:

r-arrext.p

```
DEFINE VARIABLE int_value AS INTEGER NO-UNDO EXTENT 3 INITIAL [1, 2, 3].
DEFINE VARIABLE ix        AS INTEGER NO-UNDO .
DEFINE VARIABLE tot       AS INTEGER NO-UNDO LABEL "The total is".

DO ix = 1 TO EXTENT(int_value):
  tot = tot + int_value[ix].
END.

DISPLAY tot.
```

See also

DEFINE VARIABLE statement, ENTRY function, EXTENT attribute, EXTENT statement, VAR statement

EXTENT statement

Sets the number of elements for an indeterminate array. An indeterminate array must have the extent size set before elements can be assigned into it. An indeterminate array can also be resized using the EXTENT statement. The EXTENT statement cannot be used to set or modify the size of a determinate array.

Syntax

```
EXTENT ( array ) = expression [ NO-ERROR ]
```

array

An expression that results in an ABL array.

expression

An INTEGER expression that evaluates to the extent value assigned to the array variable or parameter. The extent value can be any number between 1 and 28000, inclusive.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Note: Compound assignment operators (+=, -=, *=, /=) can be used with the EXTENT statement. For more information, see += Addition assignment operator, -= Subtraction assignment operator, *= Multiplication assignment operator, and /= Division assignment operator.

Example

The following example defines an indeterminate array variable. The size of the array variable is set multiple times using the EXTENT statement:

```
VAR CHAR[] x. // indeterminate array

MESSAGE "x has" EXTENT(x) "elements"
  VIEW-AS ALERT-BOX.  // x has ? elements

EXTENT(x) = 3.  // set size to 3 elements
x[1] = "one".
x[2] = "two".
x[3] = "three".

MESSAGE "x has" EXTENT(x) "elements" SKIP
  "x contains:" x[1] x[2] x[3]
  VIEW-AS ALERT-BOX.  // x has 3 elements
                      // x contains: one two three

EXTENT(x) = 5.  //set size to 5 elements
x[4] = "four".
x[5] = "five".

MESSAGE "x has" EXTENT(x) "elements" SKIP
  "x contains:" x[1] x[2] x[3] x[4] x[5]
  VIEW-AS ALERT-BOX.  // x has 5 elements
                      // x contains: one two three four five

EXTENT(x) = 2.  // set size to 2 elements

MESSAGE "x has" EXTENT(x) "elements" SKIP
  "x contains:" x[1] x[2]
  VIEW-AS ALERT-BOX.  // x has 2 elements
                      // x contains: one two

EXTENT(x) = 0.  // compiler error

EXTENT(x) = ?.  // deallocate the array

MESSAGE "x has" EXTENT(x) "elements"
  VIEW-AS ALERT-BOX.  // x has ? elements
```

The following example uses the += assignment operator to add 10 elements to an indeterminate array:

```
VAR CHAR[] x. // indeterminate array

MESSAGE "x has" EXTENT(x) "elements"
  VIEW-AS ALERT-BOX.  // x has ? elements

EXTENT(x) = 3.  // set size of x to 3 elements

MESSAGE "x has" EXTENT(x) "elements"
  VIEW-AS ALERT-BOX.  // x has 3 elements

EXTENT(x) += 10.  // extend size of x by 10 elements

MESSAGE "x has" EXTENT(x) "elements"
  VIEW-AS ALERT-BOX.  // x has 13 elements
```

Notes

- You can call the EXTENT statement to resize an indeterminate array. If *expression* is set to a value that is larger than the original array, the existing elements are preserved, and the additional elements are

initialized to the default value for the datatype. If *expression* is set to a smaller value, the data is truncated (elements in the array beyond the new size are lost).

- You can also use the EXTENT statement to set the size of an indeterminate array back to its initial state (with no size) by specifying the Unknown value (?) in *expression*.
- If you call the EXTENT statement attempting to set the extent size of a determinate array variable or parameter, the AVM generates a compiler error.

See also

DEFINE PARAMETER statement, DEFINE VARIABLE statement, EXTENT attribute, EXTENT function, NO-ERROR option, Parameter definition syntax, VAR statement

FILL function

Generates a character string made up of a character string that is repeated a specified number of times.

Syntax

```
FILL ( expression , repeats )
```

expression

An expression that yields a character value. This expression can contain double-byte characters.

repeats

A constant, field name, variable name, or expression with an integer value. The FILL function uses this value to repeat the *expression* you specify. If the value of *repeats* is less than or equal to 0, FILL produces a null string.

Example

This example procedure produces a bar chart that depicts each Customer's balance as a percentage of the total of all outstanding balances. The first FOR EACH block accumulates the value of balance for each Customer, producing a total balance value for all Customers. The next FOR EACH block goes through the Customer table again, figuring each Customer's balance as a percentage of the total.

r-fill.p

```
DEFINE VARIABLE fillchar AS CHARACTER NO-UNDO FORMAT "x" INITIAL "*".
DEFINE VARIABLE percentg AS INTEGER   NO-UNDO FORMAT ">>9".

FOR EACH Customer NO-LOCK:
  ACCUMULATE Customer.Balance (TOTAL).
END.

DISPLAY "Percentage of Outstanding Balance" WITH CENTERED NO-BOX.

FOR EACH Customer NO-LOCK WHERE Customer.Balance > 0:
  percentg = Customer.Balance / (ACCUM TOTAL Customer.Balance) * 100.
  FORM SKIP Customer.Name percentg LABEL "%" bar AS CHARACTER
    LABEL "  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17"
    FORMAT "x(50)" WITH NO-BOX NO-UNDERLINE USE-TEXT.
  COLOR DISPLAY BRIGHT-RED bar.
  DISPLAY Customer.Name percentg FILL(fillchar,percentg * 3) @ bar.
END.
```

The FORM statement describes the frame layout, including the name, the percentage of total balance, and a bar across the top of the frame. (The bar variable is defined on-the-fly; it has no corresponding DEFINE VARIABLE statement at the top of the procedure. It is defined in the FORM statement and has its own label and format.) The DISPLAY statement following the FORM statement displays the bar variable. If the procedure is running on UNIX or on a monochrome PC monitor, the AVM ignores the COLOR BRIGHT-RED. However, if the procedure is running on a PC with a color monitor, the bar is displayed in BRIGHT-RED (a predefined color on the PC). The final DISPLAY statement displays the bars.

The `fillchar` assignment statement sets the fill character to asterisk (*). The FILL function generates a string made up of fill characters that is the percentage of total sales multiplied by three (each percentage point uses three fill characters).

FINALLY statement

Defines a final *end block* for any ABL block, which contains code that must run regardless of an error or STOP condition. An end block is an ABL block that can occur only within another block. The block containing the end block is known as the *associated block*. End blocks must occur between the last line of executable code in the associated block and its END statement.

The purpose of a FINALLY block is to hold clean-up code that must execute regardless of what else executed in the associated block. It may include code to delete dynamic objects, write to logs, close outputs, and other routine tasks.

A FINALLY block will run on each iteration of a block, even if that iteration resulted in an error or STOP condition.

This is the syntax for the FINALLY statement and its related blocks:

Syntax

```
block-statements
    FINALLY :
        finally-logic
    END [ FINALLY ] .
[ block-end-statement ]
```

block-statements

All of the statements of an enclosing associated ABL block, including an external procedure block, though a procedure block has no *block-end-statement*. All ABL blocks other than a simple DO block (one without TRANSACTION or an ON phrase) can have a FINALLY block.

finally-logic

All statements allowed in a FINALLY block, which can include any valid ABL statement. For more information on FINALLY block execution, see the Notes for this reference entry.

block-end-statement

The END statement terminating the enclosing associated block. This applies to all ABL blocks except for a main external procedure block, which has no terminating END statement.

Examples

As shown in `r-finally01.p`, the FINALLY block executes before any flow-of-control (LEAVE, NEXT, RETRY, RETURN, or THROW) options are executed for the associated block. For iterating blocks, the FINALLY block executes after each iteration of the block.

r-finally01.p

```
DO ON ERROR UNDO, LEAVE:
  /* Since Customer 1000 does not exist, the FIND statement raises ERROR and
     execution goes to the FINALLY block before the LEAVE option executes. */
  FIND Customer WHERE CustNum = 1000.
  MESSAGE "This message never appears because of ERROR condition."
    VIEW-AS ALERT-BOX BUTTONS OK.

  FINALLY:
    MESSAGE "Inside FINALLY block." VIEW-AS ALERT-BOX BUTTONS OK.
    /* LEAVE DO block here */
  END FINALLY.
END. /* DO */

MESSAGE "Out of DO block." VIEW-AS ALERT-BOX BUTTONS OK.
```

Output:

```
** Customer record not on file (138)
Inside FINALLY block.
Out of DO block.
```

In `r-finally02.p`, after ERROR is raised, execution goes to the CATCH block and then to the FINALLY block.

r-finally02.p

```
DO ON ERROR UNDO, LEAVE:
  /* Since Customer 1000 does not exist, the FIND statement raises ERROR and
     execution goes to CATCH block. */
  FIND Customer WHERE CustNum = 1000.
  MESSAGE "This message never appears because of ERROR condition."
    VIEW-AS ALERT-BOX BUTTONS OK.

  CATCH eSysError AS Progress.Lang.SysError:
    /* Handler code for SysError condition */
    MESSAGE "Inside CATCH block." VIEW-AS ALERT-BOX BUTTONS OK.
    /* Execution goes to FINALLY before leaving DO block. */
  END CATCH.

  FINALLY:
    /* Your code */
    MESSAGE "Inside FINALLY block." VIEW-AS ALERT-BOX BUTTONS OK.
    /* LEAVE DO block here. */
  END FINALLY.
END. /* DO */

MESSAGE "Out of DO block." VIEW-AS ALERT-BOX BUTTONS OK.
```

Output:

```
Inside CATCH block.
Inside FINALLY block.
Out of DO block.
```

Notes

- There can be only one FINALLY block in any associated block. The FINALLY statement must come after all other executable statements in the associated block and before the END statement. If the associated block contains CATCH statements, the FINALLY block must come after all CATCH blocks. Note that the FINALLY statement **can** be used in a block with no CATCH blocks.
- The FINALLY block will not execute if a QUIT statement is in effect and it is not handled.
- Since it executes after an invoked CATCH block, the FINALLY block can also be used to perform common post-CATCH clean up tasks, rather than repeating common code in all the CATCH blocks present in the associated block.
- The transaction of the associated block is either complete (success) or undone (failure) when FINALLY executes. Therefore, any UNDO statement within the FINALLY block will only undo the work in the FINALLY block.
- The FINALLY block is an undoable block with implicit ON ERROR UNDO, THROW error handling, and like all blocks, FINALLY blocks implicitly throw all stop objects. You cannot explicitly override the ON ERROR directive for a FINALLY block. If a statement within the FINALLY block raises an error or STOP condition, by default, the FINALLY block will be undone, and the condition will be raised in the block that encloses the associated block of the FINALLY block. Error is not raised in the associated block. Otherwise, infinite looping could occur. To avoid having the error thrown out, you can use a CATCH block to handle the error as in other blocks.
- If the AVM detects an unhandled QUIT condition in the associated block, the FINALLY block does not run and the AVM processes the condition. If the associated block has an ON QUIT phrase, the QUIT condition

is handled and cleared by the time the AVM is ready to execute the FINALLY block, and the FINALLY block executes. See the ON QUIT phrase reference entry for a description of QUIT condition behavior and handling.

- For more information on FINALLY blocks see *ABL Error Handling.*

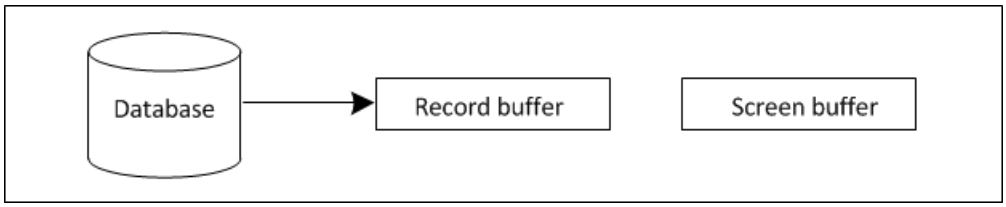
See also

CATCH statement, ON ERROR phrase, ON QUIT phrase, ON STOP phrase, RETURN statement

FIND statement

Locates a single record in a table and moves that record into a record buffer.

Data movement



Syntax

```
FIND [ FIRST | LAST | NEXT | PREV ] record
     [ constant ]
     [ OF table ]
     [ WHERE expression ]
     [ USE-INDEX index ]
     [ USING [ FRAME frame ] field
       [ AND [ FRAME frame ] field ] ...
     ]
     [ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ]
     [ NO-WAIT ]
     [ NO-PREFETCH ]
     [ NO-ERROR ]
```

You can specify the `OF`, `WHERE`, `USE-INDEX`, and `USING` options in any order.

```
FIND CURRENT record
     [ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ]
     [ NO-WAIT ]
     [ NO-ERROR ]
```

FIRST

Finds the first record in the table that meets the characteristics you might have specified with *record*. If the buffer named in the *record* was preselected in a DO or REPEAT statement, FIND locates the first record in that preselected subset of records.

LAST

Finds the last record in the table that meets the specified characteristics of the *record*. If the buffer named in the *record* was preselected in a DO or REPEAT statement, FIND locates the last record in that preselected subset of records.

NEXT

Finds the next record in the table that meets the specified characteristics of the *record*. If no record has been found, the buffer contents specified by *record* are unknown, with a ROWID equal to the Unknown value (?). If the buffer named in the *record* was preselected in a DO or REPEAT statement, FIND locates the next record in that preselected subset of records.

PREV

Finds the previous record in the table. If no record has been found, the buffer contents specified by *record* are unknown, with a ROWID equal to the Unknown value (?). If the buffer named in the *record* was preselected in a DO or REPEAT statement, FIND locates the previous record in that preselected subset of records.

CURRENT

Refetches the current record in the buffer with the specified lock status.

record

Identifies the record you want to retrieve. The *record* parameter can be a reference to a database table or a defined buffer.

constant

The value of a single component, unique, primary index for the record you want.

```
FIND Customer 1.
```

ABL converts this FIND statement with the *constant* option of 1.

```
FIND Customer WHERE Customer.CustNum = 1.
```

The CustNum field is the only component of the primary index of the Customer table. If you use the *constant* option, you must use it once in a single Record phrase, and it must precede any other options in the Record phrase.

OF *table*

Qualifies the records by relating the record to a record in another table.

```
PROMPT-FOR Order.OrderNum.
FIND Order USING OrderNum.
DISPLAY Order.
FIND Customer OF Order.
DISPLAY Customer.
```

The OF option relates the order table to the Customer table, telling the AVM to select the Customer record related to the Order record currently being used. When you use OF, all fields participate in match criteria, if an index is multi-field. The relationship is based on having a UNIQUE index in one table. ABL converts the FIND statement with the OF option to the following:

```
FIND Customer WHERE Customer.CustNum = Order.CustNum.
```

You can access related tables using WHERE, whether or not the field names of the field or fields that relate the tables have the same name.

WHERE *expression*

Qualifies the records you want to access. The *expression* is a constant, field name, variable name, or expression whose value you want to use to select records. You can use the WHERE keyword even if you do not supply an *expression.*

```
FOR EACH Customer WHERE {*}
```

The WHERE clause may not work the same way against a DataServer as it does against the OpenEdge database. Refer to the appropriate DataServer Guide (*Use the Microsoft SQL Data Server* or *Use the Oracle Data Server*) for additional information on how this feature will perform.

Note: You cannot reference an OpenEdge BLOB or CLOB field in a WHERE clause. However, in OpenEdge DataServer for MS SQL Server, a non-legacy BLOB or CLOB server data type can be mapped to the OpenEdge CHARACTER data type in a WHERE clause operand as long as the total size of the resolved WHERE clause (which might include the CHARACTER-mapped value) does not exceed 30000 bytes.

USE-INDEX *index*

Identifies the index you want to use while selecting records. If you do not use this option, the AVM selects an index to use based on the criteria specified with the WHERE, USING, OF, or *constant* options.

USING [FRAME *frame*] *field* [AND [FRAME *frame*] *field*] . . .

One or more names of fields for selecting records. The field you name in this option must have been entered previously, usually with a PROMPT-FOR statement. The field must be viewed as a fill-in or text widget.

The USING option translates into an equivalent WHERE option.

```
PROMPT-FOR Customer.CustNum.
FIND Customer USING Customer.CustNum.
```

This FIND statement is the same as the following statement:

```
FIND Customer WHERE Customer.CustNum = INPUT Customer.CustNum.
```

The CustNum field is a non-abbreviated index. However, consider this example:

```
PROMPT-FOR Customer.Name.
FIND Customer USING Customer.CustName.
```

If the name field is an abbreviated index of the Customer table, ABL converts the FIND statement with the USING option into this following statement:

```
FIND Customer WHERE Customer.Name BEGINS INPUT Customer.Name.
```

Note that field can be expanded to be FRAME *framefield.*

SHARE-LOCK

Tells the AVM to put a SHARE-LOCK on records as they are read. Other users can still read a record that is share locked, but they cannot update it. By default, the AVM puts a SHARE-LOCK on a record when it is read, and automatically puts an EXCLUSIVE-LOCK on a record when it is modified (unless the record is already EXCLUSIVE-LOCKed).

If you use the SHARE-LOCK option and the AVM tries to read a record that is EXCLUSIVE-LOCKed by another user, the AVM waits to read the record until the EXCLUSIVE-LOCK is released. The AVM displays a message to the user of that procedure, identifying the table that is in use, the user ID of the user, and the tty of the terminal using the table.

If you are using a record from a work table, the AVM disregards the SHARE-LOCK option.

EXCLUSIVE-LOCK

Tells the AVM to put an EXCLUSIVE-LOCK on records as they are read. Other users cannot read or update a record that is EXCLUSIVE-LOCKed, except by using the NO-LOCK option. They can access that record only when the EXCLUSIVE-LOCK is released. The AVM automatically puts a SHARE-LOCK on a record when it is read and automatically puts an EXCLUSIVE-LOCK on a record when it is updated.

If a record is read specifying EXCLUSIVE-LOCK, or if a lock is automatically changed to EXCLUSIVE-LOCK by an update, a user's read or update will wait if any other user has the record SHARE-LOCKed or EXCLUSIVE-LOCKed.

When a procedure tries to use a record that is EXCLUSIVE-LOCKed by another user, the AVM displays a message identifying the table that is in use, the user ID of the user, and the tty of the terminal using the table.

If you are using a record from a work table, the AVM disregards the EXCLUSIVE-LOCK option.

NO-LOCK

Tells the AVM to put no locks on records as they are read, and to read a record even if another user has it EXCLUSIVE-LOCKed.

Other users can read and update a record that is not locked. By default, the AVM puts a SHARE-LOCK on a record when it is read (unless it is using a CAN-FIND function, which defaults to NO-LOCK), and automatically puts an EXCLUSIVE-LOCK on a record when it is updated (unless the record is already EXCLUSIVE-LOCKed). A record that has been read NO-LOCK must be reread before it can be updated, as shown in this example:

```
DEFINE VARIABLE rid AS ROWID NO-UNDO.

FIND FIRST Customer NO-LOCK.
rid = ROWID(Customer).
FIND Customer WHERE ROWID(Customer) = rid EXCLUSIVE-LOCK.
```

If a procedure finds a record and it places it in a buffer using NO-LOCK and you then re-find that record using NO-LOCK, the AVM does not reread the record. Instead, it uses the copy of the record that is already stored in the buffer.

When you read records with NO-LOCK, you have no guarantee of the overall consistency of those records because another user might be in the process of changing them. For example, when a record is updated, changes to indexed fields are written immediately, but changes to other fields are deferred. In the meantime, the record is in an inconsistent state. For example, the following procedure might display a CustNum of 0 if another user's active transaction has created a record and assigned a value to the indexed field CustNum that is greater than 100:

```
FOR EACH Customer WHERE Customer.CustNum > 100 NO-LOCK:
  DISPLAY Customer.CustNum.
END.
```

If you are using a record from a work table, the AVM disregards the NO-LOCK option.

NO-WAIT

Causes FIND to return immediately and raise an error condition if the record is locked by another user (unless you use the NO-ERROR option on the same FIND statement). For example:

```
FIND Customer USING cust-name NO-ERROR NO-WAIT.
```

Without the NO-WAIT option, the AVM waits until the record is available.

The AVM ignores NO-WAIT when it is used with work tables and databases that are only accessed by a single user.

NO-PREFETCH

Specifies that only one record is sent across the network at a time. If you are accessing a remote server and do not specify this option, the AVM might send more than one record from the server to the client in each network packet.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

For the FIND statement with NO-ERROR, you can use the AVAILABLE function to test if FIND found a record.

Examples

This procedure produces a report that shows all the customers who bought a particular item, and the quantity that they bought. The procedure finds an item record, the order-lines that use that item, the order associated with each order-line, and the customer associated with each order.

r-find.p

```
REPEAT:
  PROMPT-FOR Item.ItemNum.
  FIND Item USING Item.ItemNum.
  DISPLAY Item.ItemNum Item.ItemName.
  REPEAT:
    FIND NEXT OrderLine OF Item.
    FIND Order OF OrderLine.
    FIND Customer WHERE Customer.CustNum = Order.CustNum.
    DISPLAY Customer.Name Order.OrderNum OrderLine.Qty (TOTAL).
  END.
END.
```

The FIND FIRST statement in the following procedure finds the first record with a name field value that alphabetically follows the name supplied by the user. The FIND NEXT statement uses the name index to find the next record in the table, using the name index.

r-find2.p

```
DEFINE VARIABLE start-name NO-UNDO LIKE Customer.Name.

REPEAT:
  SET start-name.
  FIND FIRST Customer WHERE Customer.Name >= start-name.
  REPEAT:
    DISPLAY Customer.Name.
    FIND NEXT Customer USE-INDEX name.
  END.
END.
```

Notes

- If a FIND statement fails, it indicates that the buffer named in *record* contains no record.
- If the AVM finds an old record in the record buffer when executing a FIND, it validates the record then writes it out. (If the record fails validation, the AVM returns an error message.) Then it clears the buffer and stores the located record in the record buffer.
- A FIND statement that does not supply FIRST, LAST, NEXT, or PREV is a unique FIND and must be able to locate, at most, one record based solely on the conditions in the expression or WHERE clause it is using.
- Fields referenced in the WHERE clause do not have to be indexed.
- WHERE conditions can include Boolean operations.

- Use caution when using conversion functions like STRING and DATE in WHERE expressions. If, for example, the -d startup parameter is used differently on the client and server sides, any conversion performed by DATE could cause different results depending on the settings.
- If a FIND NEXT or FIND PREV does not find another record, the AVM takes the end-key action. By default, this action is UNDO, LEAVE for a FOR EACH, REPEAT, or procedure block.
- See the DEFINE BUFFER statement reference entry for a description of how to use FIND on a PRESELECTed set of records.
- When you use the FIND statement, the AVM selects an index to use based on the WHERE condition or the USE-INDEX option.
- Your position in an index is established when you find a record and is only modified by subsequent record retrievals, not by CREATEs or by changing indexed field values. After a failed FIND, you cannot be certain of the cursor position.
- If you are using the FIND statement to find a record in a work table, you must use the FIRST, LAST, NEXT, or PREV option with the FIND statement.
- In a REPEAT block, if you use the FIND NEXT statement to find a record and then do an UNDO, RETRY of a block, the FIND NEXT statement reads the next record in the table, rather than the one found in the block iteration where the error occurred:

```
REPEAT:
  FIND NEXT Order.
  DISPLAY Order.
  SET Order.OrderNum.
  SET Order.OrderDate Order.PromiseDate.
END.
```

The AVM does an UNDO, RETRY if there is an error and you explicitly use the UNDO, RETRY statement, or if you press **END-ERROR** on the second or later windows interaction in a block.

Here, if you press **END-ERROR** during the second SET statement, the AVM displays the next record in the table.

If you are using a FOR EACH block to read records, and do an UNDO, RETRY during the block, you see the same record again rather than the next record.

If you want to use a REPEAT block and want to see the same record in the event of an error, use the RETRY function:

```
REPEAT:
  IF NOT RETRY THEN FIND NEXT Order.
  DISPLAY Order.
  SET Order.OrderNum.
  SET Order.OrderDate Order.PromiseDate.
END.
```

- When you use FIND NEXT or FIND PREV to find a record after updating another record, be careful not to lose your updates in case the record you want to find is unavailable.

```
FIND FIRST Customer.
REPEAT:
  UPDATE Customer.
  FIND NEXT Customer.
END.
```

  In this example, if the FIND NEXT statement fails to find the customer record, any changes made during the UPDATE statement are undone. To avoid this, use the following technique:

```
FIND FIRST Customer.
REPEAT:
  UPDATE Customer.
  FIND NEXT Customer NO-ERROR.
  IF NOT AVAILABLE Customer THEN LEAVE.
END.
```

- After you use the FIND LAST statement to find the last record in a table, the AVM positions the index cursor on that record. Any references within the same record scope to the next record fail. For example:

```
FIND LAST Customer.
RELEASE Customer.
DISPLAY AVAILABLE Customer.
REPEAT:
  FIND NEXT Customer.
  DISPLAY Customer.Name.
END.
```

  In this example, the RELEASE statement releases the last Customer record from the Customer record buffer and the following DISPLAY statement displays FALSE because the Customer record is no longer available. However, the index cursor is still positioned on that last record. Therefore, the FIND NEXT statement fails.

- If you use FIND . . . WHERE ROWID *rowid* = . . . on a PRESELECTed list of records, the temporary preselect index cursor is not reset. So, FIND NEXT does not find the record that follows record *rowid* in the preselected list. (See the DO statement and REPEAT statement reference entries for details.)
- When you use a FIND NEXT or FIND PREV statement in a subprocedure to access a record from a shared buffer, remember the following:
  - When you run an ABL procedure, the AVM creates a **cursor indicator** for each index accessed through a FIND statement in the procedure and each NEW buffer defined in the procedure. A cursor indicator serves as an anchor for index cursors associated with a table or buffer. An index cursor is attached to the cursor indicator when you enter a block of code where a record buffer is scoped. If two different indexes are used for the same record buffer within a single block of code, two index cursors are attached to the same cursor indicator. When the program control leaves the block where a record buffer is scoped, all index cursors attached to the cursor indicator are released.
  - When the AVM encounters a subprocedure in a procedure, it constant, field name, variable name, or checks through the existing index cursors before creating any other index cursors required by the statements in the subprocedure.
  - If the USE-INDEX of the FIND NEXT or FIND PREV statement in a subprocedure accesses an index cursor for a shared buffer that existed prior to the beginning of the subprocedure, the FIND NEXT or

FIND PREV statement returns the next or previous record for the shared buffer, based upon the last record found in that buffer and the USE-INDEX of the FIND statement.

- If the USE-INDEX of the FIND NEXT or FIND PREV statement in a subprocedure accesses an index cursor created for a shared buffer at the beginning of the subprocedure, the FIND NEXT or FIND PREV statement returns the first or last record for the shared buffer, based upon the USE-INDEX of the FIND statement.

- If a field or variable referenced with FIND is used in more than one frame, then the AVM uses the value in the frame most recently introduced in the procedure. To make sure you are using the appropriate frame, use the FRAME option with the FIND function to reference a particular frame.
- When a FIND statement executes, any FIND trigger defined for the table is executed.
- The FIND CURRENT statement is useful for maintaining small transaction size in updates. For an example, see the CURRENT-CHANGED function reference entry.
- FIND triggers do not execute for a FIND CURRENT statement.
- ABL does not allow a FIND statement within a FOR EACH block unless you specify a different table than the one referenced in the FOR EACH block. When you attempt to compile the following example, ABL returns the error message "FIND cannot be processed for a FOR EACH mode record":

```
FOR EACH Customer NO-LOCK:
  FIND CURRENT Customer.
END.
```

- ABL restricts the FIND statement within a PRESELECT block in the following situations:
  - You cannot specify a lock option on the FIND statement. You must specify it in the PRESELECT phrase. Attempting to compile the following example produces the error message "LOCK keyword illegal on FIND within a PRESELECT for the same table":

```
DO PRESELECT EACH Customer:
  FIND NEXT Customer NO-LOCK.
END.
```

  - You cannot specify a unique FIND or a

    FIND CURRENT

    for the same table. The following example produces the error message "Unique FIND not allowed within a PRESELECT on the same table" when you try to compile it:

```
DO PRESELECT EACH Customer:
  FIND Customer 5.
END.
```

See also

AMBIGUOUS function, AVAILABLE function, CAN-FIND function, CURRENT-CHANGED function, DEFINE BUFFER statement, ERROR-STATUS system handle, FOR statement,GET statement, LOCKED function, NEW function (record buffers), NO-ERROR option, PRESELECT phrase

FIRST function

Returns a TRUE value if the current iteration of a DO, FOR EACH, or REPEAT . . . BREAK block is the first iteration of that block.

Syntax

```
FIRST ( break-group )
```

break-group

The name of a field or expression you name in the block header with the BREAK BY option.

Example

The `r-first.p` procedure displays the order number, OrderLines on the Order, the extended price of each OrderLine, and a total order value for each Order record:

r-first.p

```
DEFINE VARIABLE order-value AS DECIMAL NO-UNDO LABEL "Order-value".

FOR EACH Order NO-LOCK:
  DISPLAY Order.OrderNum.
  FOR EACH OrderLine OF Order BREAK BY OrderLine.Qty * OrderLine.Price:
    IF FIRST(OrderLine.Qty * OrderLine.Price) THEN
      order-value = 0.
    order-value = order-value + OrderLine.Qty * OrderLine.Price.
    DISPLAY OrderLine.LineNum OrderLine.ItemNum
      OrderLine.Qty * OrderLine.Price LABEL "Extended-price".
  END.
  DISPLAY order-value.
END.
```

Because the inner FOR EACH block iterates until the AVM reads all the order-lines, the procedure must set the order-value variable to 0 each time a new order is used in that block. The FIRST function uses the (`qty * price`) expression as the *break-group* to keep track of whether or not the current iteration is the first iteration of the FOR EACH block.

See also

DO statement, FIRST-OF function, FOR statement, LAST function, LAST-OF function, REPEAT statement

FIRST-OF function

Returns a TRUE value if the current iteration of a DO, FOR EACH, or REPEAT . . . BREAK block is the first iteration for a new break group, and modifies all three block types.

Syntax

```
FIRST-OF ( break-group )
```

break-group

The name of a field or expression you name in the block header with the BREAK BY option.

Example

This procedure generates a report that lists all the item records grouped by catalog page. When the `CatPage` value changes, the procedure clears the current list of items and displays items belonging to the new catalog page. The FIRST-OF function uses the value of the `CatPage` field to determine when that value is different from the value during the last iteration.

r-firstf.p

```
FOR EACH Item BREAK BY Item.CatPage:
  IF FIRST-OF(Item.CatPage) THEN CLEAR ALL.
  DISPLAY Item.CatPage Item.ItemNum Item.ItemName.
END.
```

Note

When you calculate in a block use the BREAK option to tell the AVM to calculate when the value of certain expressions changes. The AVM uses default formatting to display the results of these calculations. To control the formatting, use the FIRST-OF function to determine the start of a break group and then change the formatting.

See also

DO statement, FIRST function, FOR statement, LAST function, LAST-OF function, REPEAT statement

FIX-CODEPAGE statement

Sets the code page of an empty LONGCHAR variable. When set to a valid code page, the code page of the specified variable is fixed and overrides any default behavior in assignment operations (including the COPY-LOB, OVERLAY, and SUBSTRING statements).

Syntax

```
FIX-CODEPAGE ( longchar ) = codepage
```

longchar

The name of a LONGCHAR variable. The variable must be set to the Unknown value (?) or the empty string (""). If the string length is greater than 0, the AVM returns an error.

codepage

A character expression that evaluates to the name of a code page. The name you specify must be a valid code page name available in `DLC/convmap.cp`. If *codepage* is the Unknown value (?), the code page of the LONGCHAR variable is not fixed.

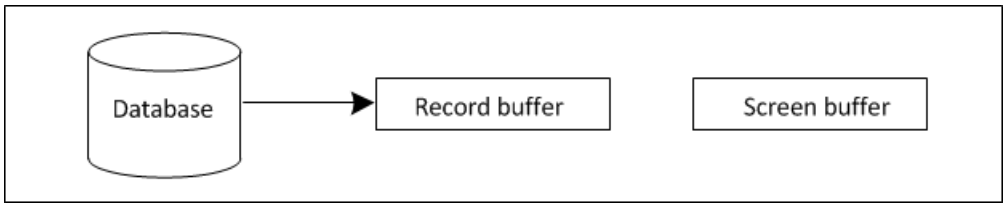
See also

GET-CODEPAGE function, IS-CODEPAGE-FIXED function

FOR statement

Starts an iterating block that reads a record from each of one or more tables at the start of each block iteration. Use an `END` statement to end a `FOR` block.

Data movement



Syntax

This is the syntax for the FOR block:

```
[ label: ]
FOR [ EACH | FIRST | LAST ]record-phrase
  [ , [ EACH | FIRST | LAST ]record-phrase ]...
  [ query-tuning-phrase ]
  [ BREAK ]
  [ BY expression[ DESCENDING ]
  | COLLATE ( string , strength[ , collation] ) [ DESCENDING ]
  ]...
  [ variable = expression1 TO expression2 [ BY k ]]
  [ WHILE expression ]
  [ TRANSACTION ]
  [ stop-after-phrase ]
  [ on-error-phrase ]
  [ on-endkey-phrase ]
  [ on-quit-phrase ]
  [ on-stop-phrase ]
  [ frame-phrase ] :

  <block of statements>

END.
```

EACH

Starts an iterating block, finding a single record on each iteration. If you do not use the EACH keyword, the *record-phrase* you use must identify exactly one record in the table.

FIRST

Uses the criteria in the *record-phrase* to find the first record in the table that meets that criteria. Note that the AVM finds the first record before any sorting. Therefore a BY option, if specified, would have no effect on the record returned.

See the Notes section for more information on using this option.

LAST

Uses the criteria in the *record-phrase* to find the last record in the table that meets that criteria. Note that the AVM finds the last record before any sorting. Therefore a BY option, if specified, would have no effect on the record returned.

See the Notes section for more information on using this option.

record-phrase

Identifies the set of records you want to retrieve. This can also be the built-in buffer name, `proc-text-buffer`, that you can use to return table rows from a stored procedure.

To use FOR EACH/FIRST/LAST to access a record in a table defined for multiple databases, you must qualify the record's table name with the database name.

This is the syntax for *record-phrase*:

```
record [ field-list ]
  [ constant ] [ OF table]
  [ USE-INDEX index | TABLE-SCAN ]
  [ USING [ FRAME frame ] field
      [ AND [ FRAME frame ] field ] . . . ]
  [ WHERE expression ]
  [ TENANT-WHERE expression [ SKIP-GROUP-DUPLICATES ] ]
  [ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ]
  [ NO-PREFETCH ]
```

Specifying multiple occurrences of *record-phrase* selects the tables using an inner join.

For more information on *record-phrase* and inner joins, see Record phrase.

query-tuning-phrase

Allows programmatic control over the execution of a DataServer query. This is the syntax for the *query-tuning-phrase*:

```
QUERY-TUNING
  (
    {[    LOOKAHEAD [ CACHE-SIZE integer ]
        | NO-LOOKAHEAD
      ]
      [ DEBUG { SQL | EXTENDED } | NO-DEBUG ]
      [ SEPARATE-CONNECTION | NO-SEPARATE-CONNECTION ]
      [ JOIN-BY-SQLDB | NO-JOIN-BY-SQLDB ]
      [ BIND-WHERE | NO-BIND-WHERE ]
      [ INDEX-HINT | NO-INDEX-HINT ]
    }
  )
```

For more information on the *query-tuning-phrase*, refer to the appropriate DataServer Guide (*Use the Oracle Data Server* or *Use the Microsoft SQL Data Server*).

BREAK

Over a series of block iterations, you might want to do some work based on whether the value of a certain field changes. This field defines a break group. For example, you might be accumulating some value, such as a total. You use the BREAK option to define `State` as the break group. For example:

```
FOR EACH Customer BREAK BY Customer.State:
  DISPLAY Customer.State Customer.Name
    Customer.CreditLimit (TOTAL BY state).
END.
```

Here, the AVM accumulates the total `CreditLimit` for all the customers in the Customer table. Each time the value of the `State` field changes, the AVM displays a subtotal of the `CreditLimit` values for customers in that state.

You can use the BREAK option anywhere in the block header, but you must also use the BY option to name a sort field.

You can use the BREAK option in conjunction with the ACCUMULATE statement and ACCUM function. For more information, see the reference entries for those language elements.

The BREAK option is intended for forward-only and read-only processing. Updating, deleting, and inserting records or repositioning using the `FIND` statement can cause unpredictable results.

`BY` *`expression`* **[** `DESCENDING` **]**

Sorts the selected records by the value of *`expression`*. If you do not use the BY option, the AVM retrieves records in the order of the index used to satisfy the *`record-phrase`* criteria, or the primary index if no criteria is given. The `DESCENDING` option sorts the records in descending order (not in the default ascending order).

Note: You cannot reference a BLOB or CLOB field in the BY option.

You can use multiple BY options to do multi-level sorting. For example:

```
FOR EACH Customer BY Customer.CreditLimit BY Customer.Name
```

Here, the Customers are sorted in order by `CreditLimit`. Within each `CreditLimit` value, Customers are sorted alphabetically by `Name`.

There is a performance benefit if an index on *`expression`* exists: BREAK BY does not have to perform the sort that is otherwise required to evaluate `FIRST`, `LAST`, `FIRST-OF`, and `LAST-OF` expressions.

Note that in a `FOR FIRST/LAST` statement, the AVM finds the first/last record before any sorting, therefore a BY option, if specified, would have no effect on the record returned.

`COLLATE (` *`string , strength`* **[** *`, collation`* **]** `)` **[** `DESCENDING` **]**

Generates the collation value of a string after applying a particular strength, and optionally, a particular collation. The `DESCENDING` option sorts the records in descending order (not in default ascending order).

string

A CHARACTER expression that evaluates to the *string* whose collation value you want to generate.

strength

A CHARACTER expression that evaluates to an ABL comparison strength or an International Components for Unicode (ICU) comparison strength.

The ABL comparison strengths include:

- **RAW** — Generates a collation value for the string based on its binary value.
- **CASE-SENSITIVE** — Generates a case-sensitive collation value for the string based on a particular collation. If you specify this strength with an ICU collation, the AVM applies the ICU TERTIARY strength.
- **CASE-INSENSITIVE** — Generates a case-insensitive collation value for the string based on a particular collation. If you specify this strength with an ICU collation, the AVM applies the ICU SECONDARY strength.
- **CAPS** — Generates a collation value for the string based on its binary value after converting any lowercase letters in the string to uppercase letters, based on the settings of the Internal Code Page (`-cpinternal`) and Case Table (`-cpcase`) startup parameters.

The ICU comparison strengths include:

- **PRIMARY** — Generates a collation value for the base characters in the string.
- **SECONDARY** — Generates a collation value for the base characters and any diacritical marks in the string.
- **TERTIARY** — Generates a case-sensitive collation value for the base characters and any diacritical marks in the string.
- **QUATERNARY** — Generates a case-sensitive collation value for the base characters and any diacritical marks in the string, and distinguishes words with and without punctuation. ICU uses this strength to distinguish between Hiragana and Katakana when applied with the ICU-JA (Japanese) collation. Otherwise, it is the same as TERTIARY.

Note: Use ICU comparison strengths only with ICU collations.

collation

A CHARACTER expression that evaluates to the name of an ABL collation table or ICU collation. If *collation* does not appear, COLLATE uses the collation table of the client.

The AVM reports an error and stops execution if one of the following occurs:

- *strength* does not evaluate to a valid value.
- *collation* does not evaluate to a collation table residing in the `convmap.cp` file.
- *collation* evaluates to a collation table that is not defined for the code page corresponding to the `-cpinternal` startup parameter.

variable = *expression1* TO *expression2* [BY *k*]

Identifies the name of a field or variable whose value you are incrementing in a loop. The *expression1* is the starting value for *variable* on the first iteration of the loop. The *k* is the amount to add to *variable* after each iteration and must be a constant. It (*k*) defaults to 1. The *variable, expression1*, and *expression2* parameters must be integers.

When *variable* exceeds *expression2* (or is less than *expression2* if *k* is negative) the loop ends. Since *expression1* is compared to *expression2* at the start of the first iteration of the block, the block can be executed 0 times. The AVM re-evaluates *expression2* on each iteration of the block.

WHILE *expression*

Indicates the condition in which you want the FOR EACH block to continue processing the statements within it. Using the WHILE *expression* option causes the block to iterate as long as the condition specified by the expression is TRUE or the AVM reaches the end of the index it is scanning, whichever comes first. The expression is any combination of constants, operators, field names, and variable names that yield a logical value.

TRANSACTION

Identifies the FOR EACH block as a system transaction block. The AVM starts a system transaction for each iteration of a transaction block if there is not already an active system transaction. See *ABL Essentials* for more information on transactions.

stop-after-phrase

Specifies a time-out value (in seconds) for the block. This is the syntax for the STOP-AFTER phrase:

```
STOP-AFTER time-limit
```

For more information see STOP-AFTER phrase.

on-error-phrase

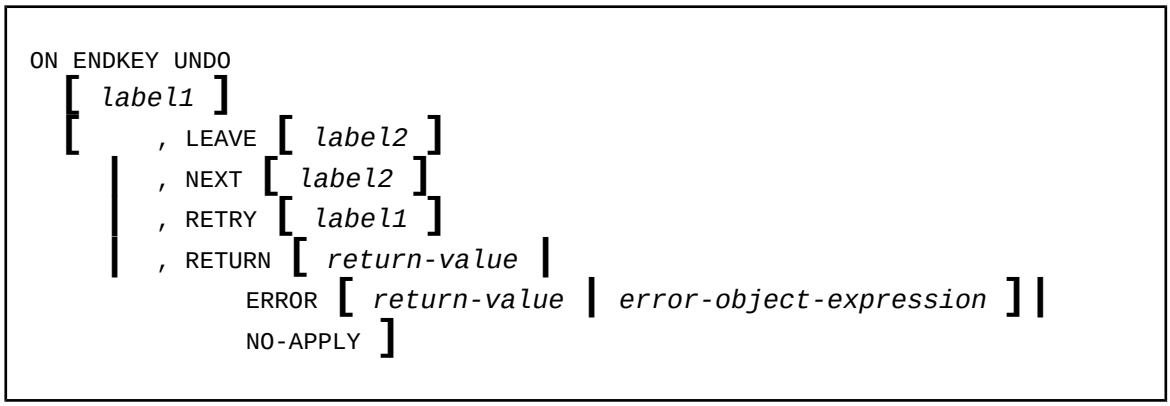
Describes the processing that takes place when there is an error during a block. This is the syntax for the ON ERROR phrase:

```
ON ERROR UNDO
  [ label1 ]
  [      , LEAVE [ label2 ]
       | , NEXT [ label2 ]
       | , RETRY [ label1 ]
       | , RETURN [ return-value |
                ERROR [ return-value | error-object-expression ] |
                NO-APPLY ]
       | , THROW
  ]
```

For more information, see the ON ERROR phrase.

on-endkey-phrase

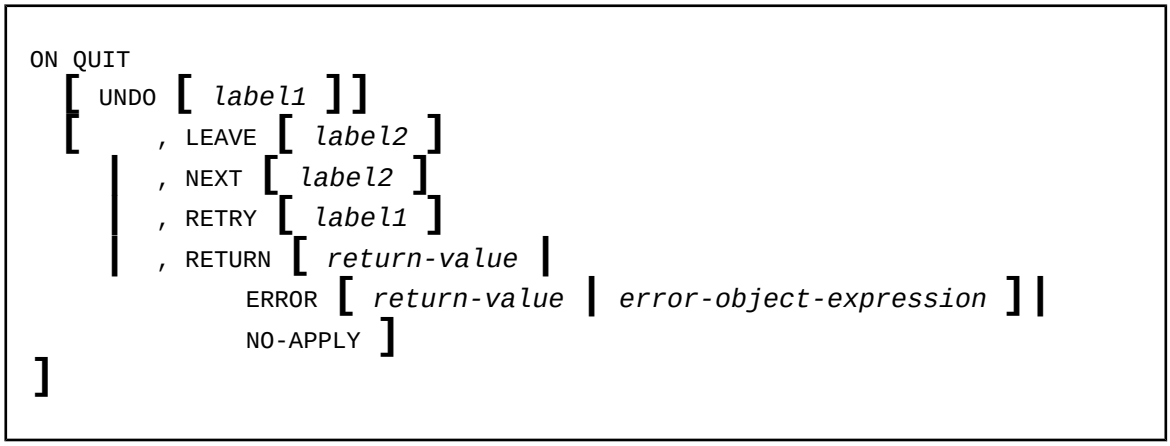
Describes the processing that takes place when the **ENDKEY** condition occurs during a block. This is the syntax for the `ON ENDKEY` phrase:



For more information, see the ON ENDKEY phrase.

on-quit-phrase

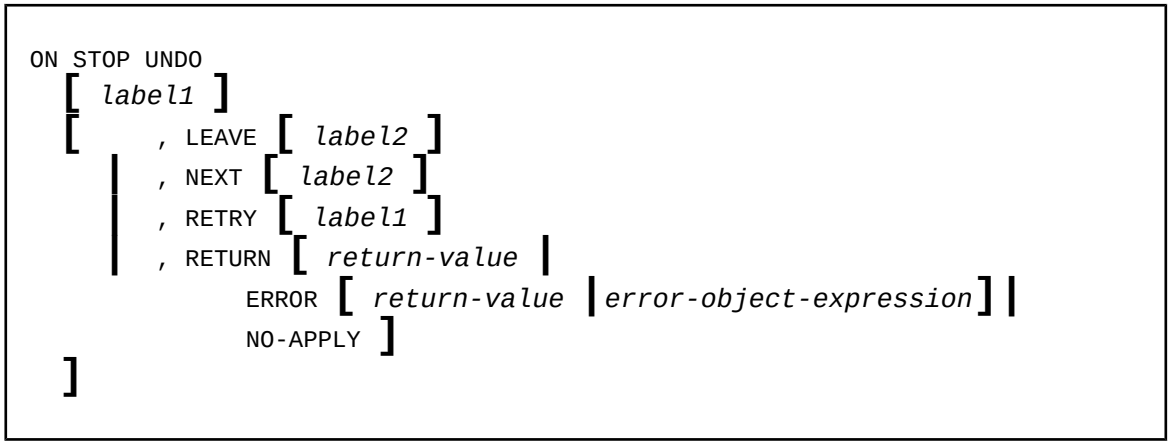
Describes the processing that takes place when a `QUIT` statement is executed during a block. This is the syntax for the `ON QUIT` phrase:



For more information, see the ON QUIT phrase.

on-stop-phrase

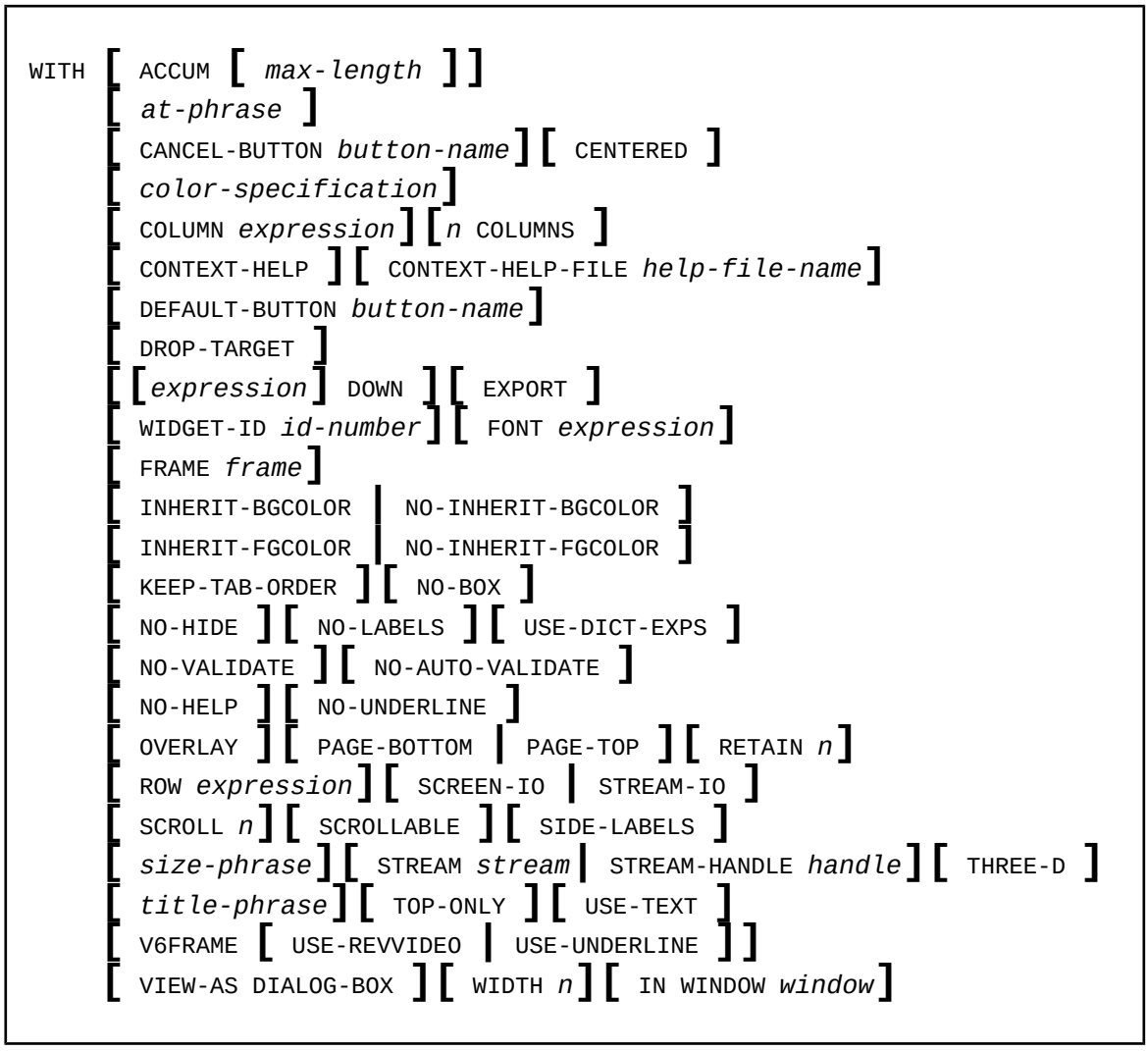
Describes the processing that takes place when the STOP condition occurs during a block. This is the syntax for the ON STOP phrase:



For more information, see the ON STOP phrase.

frame-phrase

Specifies the overall layout and processing properties of a frame. This is the syntax for the frame phrase:



For more information on *frame-phrase*, see the Frame phrase.

Examples

The following example code displays the Customer with the lowest `CreditLimit`. Note that you cannot use a `FOR FIRST` statement to do this because the AVM finds the first record before any sorting, Therefore a `BY` option, if specified, would have no effect on the record returned. Instead you can use a `FOR EACH` statement and exit the loop after displaying the first record.

```
FOR EACH Customer NO-LOCK BY Customer.CreditLimit:
  DISPLAY Customer.
  LEAVE.
END.
```

The following code reads Customer records that have a CustNum less than 12, sorting the records in order by State before displaying them:

```
FOR EACH Customer NO-LOCK WHERE Customer.CustNum < 12 BY Customer.State:
  DISPLAY Customer.CustNum Customer.Name Customer.City Customer.State.
END.
```

The following example code gets information from four related tables (Customer, Order, OrderLine, and Item) and displays some information from each. Before displaying the information, the FOR EACH statement sorts it in order by the PromiseDate field, then, within that field, in order by CustNum. Within the CustNum field, the data is sorted by the LineNum field.

```
FOR EACH Customer NO-LOCK, EACH Order OF Customer NO-LOCK,
  EACH OrderLine OF Order NO-LOCK, Item OF OrderLine NO-LOCK
  BY Order.PromiseDate BY Customer.CustNum BY OrderLine.LineNum:
  DISPLAY Order.PromiseDate Customer.CustNum Order.OrderNum
    OrderLine.LineNum Item.ItemNum Item.ItemName.
END.
```

The following code uses the LAST option to display information on the last Order of each Customer:

```
FOR EACH Customer NO-LOCK, LAST Order OF Customer:
  DISPLAY Customer.CustNum Customer.Name Order.OrderNum Order.OrderDate
    Order.Instructions.
  PAUSE 1 NO-MESSAGE.
  Order.Instructions = "Last order".
  DISPLAY Order.Instruction.
END.
```

Notes

- At compile time, ABL determines which index or indexes to use for retrieving records from a table, based on the conditions in the *record-phrase*. For compatibility with Progress Version 6 or earlier, you can force ABL to use only one index by specifying the USE-INDEX option or by using the Version 6 Query (-v6q) parameter.
- If you specify the -v6q startup parameter, an index component is involved in an equality match if it is used in the *record-phrase* conditions in the following form:

  ```
  field = expression
  ```

  Where the *expression* is independent of any fields in the table that the index is being selected from. A condition involving OF and USING are equivalent to this form. A field is involved in a range match if it is used in a condition of this form:

  ```
  field [ < | <= | > | >= | BEGINS ]expression
  ```

  The BEGINS operator translates into two range matches for a field.

An equality or range match is considered active if the equality or range condition stands on its own or is related to other conditions solely through the AND operator (for example, not through OR or NOT).

A field is involved in a sort match if it is used in a BY option of this form:

```
BY field[ DESCENDING ]
```

- If you specify the -v6q startup parameter, the following list describes the rules the OpenEdge database manager uses to choose an index for an OpenEdge database:
  - If you specify the record by ROWID, the AVM accesses the record directly without using an index.
  - If you use the USE-INDEX option, in the *record-phrase*, the AVM uses the index you name in that option.
  - For each index in the table, the ABL compiler looks at each index component in turn and counts the number of active equality, range, and sort matches. ABL ignores the counts for any components of an index that occur after a component that has no active equality match. ABL compares the results of this count and selects the best index. ABL uses the following order to determine the better of any two indexes:
    1. If one index is unique and all of its components are involved in active equality matches and the other index is not unique, or if not all of its components are involved in active equality matches, ABL chooses the former of the two.
    2. Select the index with more active equality matches.
    3. Select the index with more active range matches.
    4. Select the index with more active sort matches.
    5. Select the index that is the primary index.
    6. Select the first index alphabetically by index name.

- If you specify the -v6q startup parameter, the AVM might have to scan all the records in the index to find those meeting the conditions, or it might have to examine only a subset of the records. This latter case is called bracketing the index and results in more efficient access. Having selected an index as previously described, the ABL compiler examines each component as follows to see if the index can be bracketed:
  - If the component has an active equality match, ABL can bracket it, and it examines the next component for possible bracketing.
  - If the component has an active range match, ABL can bracket it, but it does not examine the remaining components for possible bracketing.
  - If the component does not have an active equality match or an active range match, ABL does not examine the remaining components for bracketing.

- If you specify the v6q parameter, any conditions you specify in the *record-phrase* that are not involved in bracketing the selected index are applied to the fields in the record itself to determine if the record meets the overall *record-phrase* criteria. For example, assume that the f table has fields a, b, and c involved in two indexes:
  - Primary, unique index (I1) on a, b, and c
  - Secondary non-unique index (I2) on c

  The following table shows the index ABL selects and the bracketed part of the index for various *record-phrases*.

Table 38: Progress Version 6 index selection examples

Record phrase	Index selected	Bracketing on
`f WHERE a = 3 AND b = 2 AND c = 3`	I1	a + b + c
`f WHERE a = 3`	I1	a
`f WHERE c = 1`	I2	c
`f WHERE a = 3 AND b > 7 AND c = 3`	I1	a + b
`f WHERE a = 3 AND c = 4`	I1	a
`f WHERE b = 5`	I1	None of the fields[34]
`f WHERE a = 1 OR b >5`	I1	None of the fields[34]
`f WHERE (a >= a1 AND a <= a2)` `OR (a1=0)`	I1	None of the fields[35]

`f WHERE a >= (IF a1 NE 0 THEN a1`
`ELSE -99999999) AND`
`a <= (IF a1 NE 0 THEN a2`
`ELSE +99999999)`
	I1	a[35]

- The FIRST and LAST keywords are especially useful when you are sorting records in a table in which you want to display information. Often, several related records exist in a related table, but you only want to display the first or last related record from that table in the sort. You can use FIRST or LAST in these cases.

  Suppose you were interested in displaying the date when each Customer first placed an order. This procedure displays the Customer number and date of the first Order:

```
FOR EACH Customer NO-LOCK, FIRST Order OF Customer NO-LOCK:
  DISPLAY Order.CustNum Order.OrderDate.
END.
```

[34] In this case, ABL must look at all of the records to determine which meet the specified criteria.

[35] The two record phrases in these examples are almost identical in effect, but the one using the OR operator to connect conditions is much less efficient in its use of the selected index.

The following example code displays the last Order Line of every Order, sorted by the Price of the Item and by the Promised Date of the Order:

```
DISPLAY "Show the last order-line of each order," SKIP
  "sorted by the item's price and the" SKIP
  "promised date of the order." WITH CENTERED.

FOR EACH Order NO-LOCK, LAST OrderLine OF Order NO-LOCK,
  Item OF OrderLine NO-LOCK BY Item.Price BY Order.PromiseDate:
  DISPLAY Order.OrderNum OrderLine.LineNum Item.ItemNum OrderLine.Price
    Order.PromiseDate WITH TITLE "For FIRST/LAST" CENTERED.
END.
```

- If you want the AVM to use a specific index, you must specify the first component of that index in the record phrase of the FOR statement.
- You cannot reference a BLOB or CLOB field in a WHERE clause.
- For SpeedScript, the *on-endkey-phrase* and the *on-quit-phrase* do not apply.

See also

CATCH statement, FINALLY statement, FIND statement, Frame phrase, ON ENDKEY phrase, ON ERROR phrase, ON QUIT phrase, ON STOP phrase, Record phrase, STOP-AFTER phrase

---

FORM statement

Defines the layout and certain processing attributes of a frame for use within a single procedure. If the frame has not been previously scoped, the FORM statement scopes it to the current block. Use the FORM statement if you want to describe a frame in a single statement rather than let ABL construct the frame based on individual data handling statements in a block. You can use the FORM statement to describe a layout for a data iteration and the frame header or background.

Syntax

```
FORM
  [ form-item . . . ]
  [ { HEADER | BACKGROUND } head-item . . . ]
  [ frame-phrase ]
```

```
FORM record [ EXCEPT field . . . ] [ frame-phrase ]
```

form-item

Specifies a field-level widget or value to display in the frame, or a SPACE or SKIP directive. The data specified by all form items are owned by a single field group, duplicated for each data iteration in the frame.

This is the syntax for *form-item*:

```
field [ format-phrase ]
```

```
constant
  [ at-phrase | TO n ]
  [ BGCOLOR expression ]
  [ DCOLOR expression ]
  [ FGCOLOR expression ]
  [ FONT expression ]
  [ PFCOLOR expression ]
  [ VIEW-AS TEXT ]
  [ WIDGET-ID id-number ]
```

```
SPACE [ ( n ) ]
```

```
SKIP [ ( n ) ]
```

field

A reference to a field or variable to be displayed in the frame. This value cannot be an expression or a frame. To specify a child frame, you must first define the parent and child frames, then assign the FRAME attribute of the child frame to the handle of the parent frame. The child frame is assigned to the same field group as other form items.

format-phrase

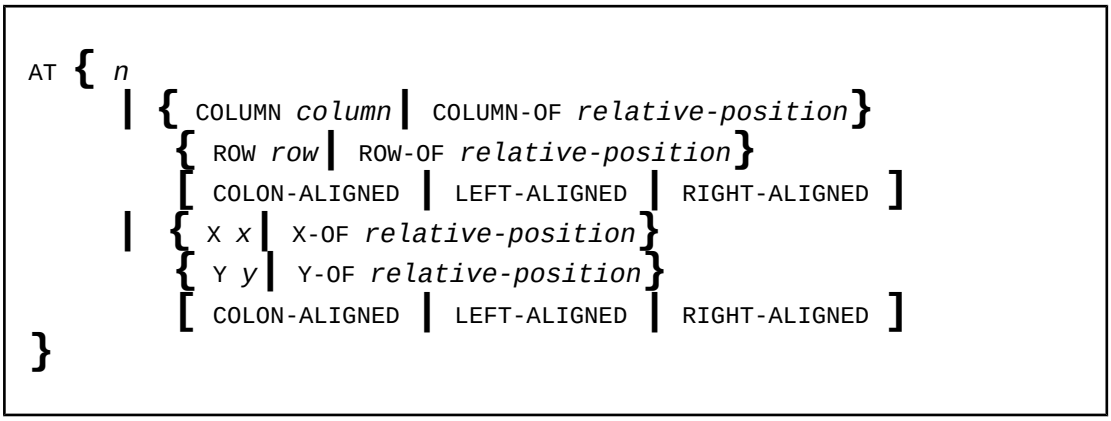
Specifies one or more frame attributes for a field or variable. For more information on *format-phrase*, see the Format phrase reference entry.

constant

A constant value.

at-phrase

Specifies the location of a value within the frame. The AT phrase does not left justify the data; it simply indicates the placement of the data area. This is the syntax for the AT phrase:



For more information, see the AT phrase reference entry.

TO *n*

The number (*n*) of the column in which you want the display to end. The TO option does not right justify the data; it simply indicates the placement of the data area.

BGCOLOR *expression*

Specifies the background color of the form item in graphical interfaces. This option is ignored in character interfaces.

DCOLOR *expression*

Specifies the display color of the form item in character interfaces. This option is ignored in graphical interfaces.

FGCOLOR *expression*

Specifies the foreground color of the form item in graphical interfaces. This option is ignored in character interfaces.

FONT *expression*

Specifies the font of the form item.

PFCOLOR *expression*

Specifies the prompt color of the form item in character interfaces. This option is ignored in graphical interfaces.

VIEW-AS TEXT

Specifies that the form item be displayed as a TEXT widget rather than as a FILL-IN widget.

WIDGET-ID *id-number*

Specifies a widget ID for a field-level widget or value to display in a frame. The value of *id-number* must be an expression that evaluates to an even integer value between 2 and 65534, inclusive, and must be unique across all widget IDs in the window or dialog box.

If you specify an invalid ID, the compiler displays an error message. This option is supported in graphical interfaces only, and only in Windows.

SPACE (*n*)

Identifies the number (*n*) of blank spaces to insert after the displayed expression. The *n* can be 0. If the number of spaces you specify is more than the spaces left on the current line of the frame, the AVM starts a new line and discards extra spaces. If you do not use this option or you do not use *n*, the AVM inserts one space between items in the frame.

SKIP (*n*)

Identifies the number (*n*) of blank lines to insert after the displayed expression. The number of blank lines can be can be 0. If you do not use this option, the AVM does not skip a line between expressions unless the expressions do not fit on one line. If you use the SKIP option but do not specify *n,* or if *n* is 0, the AVM starts a new line unless it is already at the beginning of a new line.

record

Represents the name of the record you want to display. Naming a record is shorthand for listing each field individually, as a form item.

EXCEPT *field* . . .

Tells the AVM to display all the fields in the frame except those fields listed in the EXCEPT phrase.

HEADER

Tells the AVM to place the following items in a header section at the top of the frame in a separate field group from all other data. In addition to fields, variables, and constants, the frame header can contain expressions, images, and rectangles. The AVM reevaluates these expressions each time it displays the frame.

When you use the FORM statement with the HEADER option, ABL disregards Data Dictionary field labels for fields you name in the FORM statement. Use character strings to specify labels for fields you name in the frame header.

BACKGROUND

Specifies that any following frame items display in the frame background, behind the data and header in a separate field group. Typically, this option is used to display images or rectangles behind the data.

head-item

A description of a value to be displayed in the frame header or background, or a SPACE or SKIP directive. This is the syntax for head-item:

```
expression [ format-phrase ]
```

```
constant
   [ at-phrase | TO n ]
   [ BGCOLOR expression ]
   [ DCOLOR expression ]
   [ FGCOLOR expression ]
   [ FONT expression ]
   [ VIEW-AS TEXT ]
   [ WIDGET-ID id-number ]
```

```
SPACE [ ( n ) ]
```

```
SKIP [ ( n ) ]
```

This is exactly the same as the syntax for a *form-item*, except that a *head-item* can be an expression and does not include the PFCOLOR option. If you use an expression in a HEADER or BACKGROUND phrase, the expression is evaluated each time the frame is viewed. If you give the PAGE-TOP or PAGE-BOTTOM option for the frame, the expression is evaluated for each page. This allows you, for example, to include a reference to the PAGE-NUMBER function in the frame header.

Note: If *head-item* is an expression, any option of the *format-phrase* may be used with it; if *head-item* is a constant, only the AT phrase, TO, BGCOLOR, DCOLOR, FGCOLOR, FONT, VIEW-AS TEXT, and WIDGET-ID options are allowed.

---

frame-phrase

Specifies frame options for the frame associated with the FORM statement. For more information on *frame-phrase* options, see the Frame phrase reference entry.

Examples

This procedure lets the user update information on a specific Customer. The FORM statement describes a very specific layout for the UPDATE statement to use.

r-form.p

```
REPEAT FOR Customer:
  FORM
    Customer.Name COLON 10 Customer.Phone COLON 50
    Customer.Address COLON 10 Customer.SalesRep COLON 50 SKIP
    Customer.City COLON 10 NO-LABEL Customer.State NO-LABEL
       Customer.PostalCode NO-LABEL
    WITH SIDE-LABELS 1 DOWN CENTERED.
  PROMPT-FOR Customer.CustNum WITH FRAME cnum SIDE-LABELS CENTERED.
  FIND Customer USING Customer.CustNum.
  UPDATE Customer.Name Customer.Address Customer.City Customer.State
    Customer.PostalCode Customer.Phone Customer.SalesRep.
END.
```

When you use the FORM statement to control the order in which fields appear on the screen, remember that this order is independent of the order in which the AVM processes the fields during data entry.

In the example, the above FORM statement displays the customer name first and the phone number second. But the UPDATE statement specifies the phone number after the name, address, city, state, and postal-code. The fields are displayed as described in the FORM statement, but the tab order is determined by the UPDATE statement.

The following example uses the HEADER option:

r-eval.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO FORMAT ">9".

FORM HEADER "This is the header - ix is" ix
  WITH FRAME a ROW ix COLUMN ix ix DOWN.

DO ix = 1 TO 8 WITH FRAME a
  DISPLAY ix.
  PAUSE.
END.
```

The FORM statement defines a HEADER frame that consists of the text "This is the header - ix is" and the value of the variable ix. In addition, it also specifies a screen location where the header is displayed. The FORM statement does not bring the header frame into view.

On the first iteration of the DO block, the DISPLAY statement brings the frame into view. On the second iteration of the DO block, the frame is already in view (it was not hidden during the first iteration), so the header of the frame is not re-evaluated. Thus, the new value of ix is not reflected in the header portion of the frame, and you do not see the new value of ix in the header. You also do not see the position of the frame on the screen change.

In contrast, look at this modified version of the procedure:

r-eval2.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO FORMAT ">9".

FORM HEADER "This is the header - ix is" ix
  WITH FRAME a ROW ix COLUMN ix ix DOWN.

DO ix = 1 TO 8 WITH FRAME a:
  DISPLAY ix.
  HIDE FRAME a.
END.
```

On the first iteration of the DO block, the DISPLAY statement displays the frame. The HIDE statement removes the frame from the window. Therefore, on the second iteration of the DO block, the DISPLAY statement redisplays the frame. The AVM re-evaluates the header of the frame each time the frame is redisplayed. Therefore, the header of the frame reflects the change to i, and the position of the frame in the window also changes.

Notes

- When you use any of the statements that access the screen, you can name a frame or use the default frame for the block where the statements appears.
- When ABL compiles a procedure, it makes a top-to-bottom pass of the procedure to design all the frames for that procedure, including those referenced in FORM statements. ABL adds field and format attributes as it goes through the procedure.
- If you have enabled application-defined widget IDs in your ABL GUI application, by specifying the Use Widget ID (`-usewidgetid`) startup parameter, then the AVM uses the value specified in the WIDGET-ID option to set the WIDGET-ID attribute for this widget when it creates the widget at run time, instead of using the widget ID it normally generates by default. If you have not enabled application-defined widget IDs, then the AVM ignores this option setting at run time.

  For more information about the WIDGET-ID attribute, see its reference entry in the Handle Attributes and Methods Reference. For more information about the Use Widget ID (`-usewidgetid`) startup parameter, see *Startup Command and Parameter Reference*.
- If you use a single qualified identifier with the FORM statement, the compiler first interprets the reference as *`dbname.tablename`*. If the compiler cannot resolve the reference as *`dbname.tablename`*, it tries to resolve it as *`tablename.fieldname`*. When naming fields in a FORM statement, you must use table names that are different from field names to avoid ambiguous references. See the Record phrase reference entry for more information.
- To use the FORM statement to display a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the *`record`* definition in the Record phrase reference entry for more information.
- If you define a frame to use as a DDE frame, you must realize the frame (display it) before using it as a conversation end-point. If you want the DDE frame to remain invisible during its use in a DDE conversation, set its HIDDEN attribute to TRUE after realizing the frame. For information on DDE frames, see *OpenEdge Programming Interfaces*.

See also

DEFINE FRAME statement, Format phrase, Frame phrase

Format phrase

Specifies one or more attributes for a widget.

Syntax

```
[ at-phrase ]
[ AS datatype | LIKE field ]
[ AUTO-RETURN ]
[ BGCOLOR expression ]
[ BLANK ]
[ COLON n | TO n ]
[ COLUMN-LABEL label ]
[ DEBLANK ]
[ DCOLOR expression ]
[ DISABLE-AUTO-ZAP ]
[ FGCOLOR expression ]
[ FONT expression ]
[ FORMAT string ]
[ HELP string ]
[ LABEL label [ , label ]... | NO-LABELS ]
[ NO-TAB-STOP ]
[ PFCOLOR expression ]
[ VALIDATE ( condition , msg-expression ) ]
[ view-as-phrase ]
[ WIDGET-ID id-number ]
```

at-phrase

The column, row and column, or x and y pixel location you want the display to start. The AT option does not left justify the data; it simply indicates the placement of the data area.

```
AT { n
   | { COLUMN column | COLUMN-OF relative-position }
     { ROW row | ROW-OF relative-position }
     [ COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED ]
   | { X x | X-OF relative-position }
     { Y y | Y-OF relative-position }
     [ COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED ]
  }
```

See the AT phrase reference entry for more information.

AS *datatype*

Creates a frame field and variable with the data type you specify. This is useful for defining display positions in a frame for use with DISPLAY @ *field*.

LIKE *field*

Creates a frame field and variable with the same definition as *field*.

The LIKE option in a DEFINE VARIABLE statement, DEFINE WORK-TABLE statement, or Format phrase requires that a particular database is connected. Since you can start up an ABL application session without connecting to a database, use the LIKE option with caution.

AUTO-RETURN

Causes the AVM to automatically move out of a field as if you pressed **RETURN**. When you enter the last character in the field, the AVM automatically moves out of the field. If this happens on the last field of a data entry statement, the AVM functions as if you pressed **GO**.

For the purposes of AUTO-RETURN, entering leading zeros in a numeric field does not count as filling the field. For example, suppose you define a numeric field as follows:

```
DEFINE VARIABLE x AS INTEGER NO-UNDO FORMAT "99".

SET x AUTO-RETURN.
```

If you enter a 09 into the field, the AVM does not AUTO-RETURN. To get the AUTO-RETURN behavior in this situation, define the field as CHARACTER with a format of "99".

BGCOLOR *expression*

Specifies the background color of the widget in graphical interfaces. This option is ignored in character interfaces.

BLANK

Displays blanks for the field you are displaying or entering. This is useful for entering passwords.

COLON *n*

The number (*n*) of the column in which you want the colon of the label to appear. Use this option with SIDE-LABEL frames where the labels are placed to the left of the data and are separated from the data with a colon. For code compiled with the `-nocolon` startup parameter in effect, the last character in the label appears in the specified column. See the *Startup Command and Parameter Reference* for more information on `-nocolon` .

TO *n*

The number (*n*) of the column in which you want to end the display. The TO option does not right justify the data; it indicates the placement of the data area.

COLUMN-LABEL *label*

Names the label you want to display above the field. If you want the label to use more than one line (stacked labels), use an exclamation point (!) in the label to indicate where to break the line. For example:

r-colbl.p

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.Name COLUMN-LABEL "Customer!Name"
    Customer.SalesRep COLUMN-LABEL "Name of!Sales!Representative".
END.
```

The AVM does not display column labels if you use the SIDE-LABELS or the NO-LABELS option with the Frame phrase.

You must enclose the label string in quotation marks. If you want to use the exclamation point (!) as one of the characters in a column label, use two exclamation points (`! !`).

`DEBLANK`

Removes leading blanks (for use on input character fields only). Leading blanks in the value before input are not removed unless the user changes the value.

`DCOLOR` *`expression`*

Specifies the display color of the widget in character interfaces. This attribute is ignored in graphical interfaces.

`DISABLE-AUTO-ZAP`

Specifies whether the value of the AUTO-ZAP attribute will be ignored. See the AUTO-ZAP attribute reference entry. This option only applies to fill-ins.

The following example defines a frame with two fill-ins, both of which specify the DISABLE-AUTO-ZAP option:

```
DEFINE FRAME frame-a
  fill-in-1 DISABLE-AUTO-ZAP
  fill-in-2 DISABLE-AUTO-ZAP
  button-1
  WITH THREE-D SIDE-LABELS.
```

`FGCOLOR` *`expression`*

Specifies the foreground color of the widget in graphical interfaces. This option is ignored in character interfaces.

`FONT` *`expression`*

Specifies the font of the widget.

`FORMAT` *`string`*

Represents the format in which you want to display the *`expression`*. You must enclose string in quotation marks (""). If you do not use the FORMAT option, ABL uses the defaults shown in the following table.

Table 39: Default display formats

Type of expression	Default format
Field	Format from Dictionary
Variable	Format from variable definition
Constant character	Length of character string
Other	Default format for the data type of the expression

The following table lists the default formats for the Other expression.

Table 40: Default data type display formats

Data type	Default display format
CHARACTER	x(8)
CLASS[36]	N/A
DATE	99/99/99
DATETIME	99/99/9999 HH:MM:SS.SSS
DATETIME-TZ	99/99/9999 HH:MM:SS.SSS+HH:MM
DECIMAL	->>,>>9.99
HANDLE[37]	>>>>>>9
INT64	->,>>>,>>9
INTEGER	->,>>>,>>9
LOGICAL	yes/no
LONGCHAR[38]	See footnote.[38]
MEMPTR[39]	See footnote [39].
RAW[39]	See footnote [39].

[36] If you display a class instance using the MESSAGE statement, ABL automatically invokes the ToString( ) method (Object) (provided by the Progress.Lang.Object class) on the object reference. To display a class instance in a frame (for example, using the DISPLAY statement), you must first explicitly convert the object reference to a displayable type using the INT64 function, the INTEGER function, the STRING function, or the ToString( ) method (Object).

[37] To display a HANDLE, you must first convert it using either the INT64 or INTEGER function and display the result.

[38] You can only display a LONGCHAR using the VIEW-AS EDITOR LARGE phrase.

[39] You cannot display a MEMPTR, RAW, or ROWID value directly. However, you can convert it to a character string representation using the STRING function and display the result. A ROWID value converts to a hexadecimal string, "0x*hexdigits*", where *hexdigits* is any number of characters "0" through "9" and "A" through "F". A MEMPTR or RAW value converts to decimal integer string.

Data type	Default display format
RECID	>>>>>>9
ROWID[39]	See footnote [39].

You can use the FORMAT option with the UPDATE and SET statements to store a character string that is longer than the field length you define in the Data Dictionary or in a DEFINE VARIABLE statement. This is possible because ABL stores data in variable-length fields.

```
DEFINE VARIABLE mychar AS CHARACTER NO-UNDO FORMAT "x(3)".

UPDATE mychar FORMAT "x(8)".
```

You can also use the ASSIGN statement to store data in a field or variable that is longer than the predefined format of that field or variable.

```
mychar = "abcdefgh".
```

However, the Data Dictionary load program only loads character data that is **no longer** than the format you defined in the Dictionary. For more information on data formats, see *ABL Essentials*.

HELP *string*

Represents a character string that you want to display whenever the user enters the frame field for the field or variable. When the user leaves the frame field, the AVM removes the help string from the message area. You must enclose the string in quotation marks (""").

If the input source is not the terminal, the AVM disregards any HELP options.

LABEL *label* [, *label*] . . .

Represents a character string that you want to use as a label for a field, variable, or expression. You must enclose the string in quotation marks (""). The following table shows the order ABL uses to determine the label for a field, variable, or expression.

Table 41: Determining labels

	LABEL string	Dictionary label	Field name	LIKE field	Variable name
Field	1	2	3	N/A	N/A
Variable	1	N/A	N/A	2	3
Expression	1	N/A	N/A	N/A	N/A

Note: If you use side labels, Windows allows a user to transfer focus to field-level widgets by pressing **ALT** and one of the letters in the widget's label. This is called a mnemonic. Specify the letter by preceding it with an ampersand (&) when specifying the LABEL option. Ending a label with an ampersand might produce undesired behavior. If you want a literal ampersand within a label, enter two ampersands (&&) in *label*. If you specify more than one widget with the same mnemonic, the AVM transfers focus to each of these in tab order when you make a selection.

`NO-LABELS`

Prevents the AVM from displaying a label for a field, variable, or expression.

`NO-TAB-STOP`

Specifies that the widget is not in its parent frame's tab order.

The following example shows defining a frame with two fill-ins, both of which have the NO-TAB-STOP option specified:

```
DEFINE FRAME frame-a
  fill-in-1
  fill-in-2
  button-1 NO-TAB-STOP
  WITH THREE-D SIDE-LABELS.
```

See the TAB-STOP attribute reference entry for related information.

`PFCOLOR` *`expression`*

Specifies the prompt color of the widget in character interfaces. This attribute is ignored in graphical interfaces.

`VALIDATE (` *`condition, msg-expression`* `)`

Specifies a value that you want to validate against the data entered into a screen field or variable. The *`condition`* is a Boolean expression (a constant, field name, variable name, or expression) whose value is TRUE or FALSE.

When you use the VALIDATE option to validate a specific field, any reference to that field in *`condition`* is assumed to be an input field. For example, in the following statement, the AVM assumes the `PromiseDate` field is an input field:

```
SET Order.OrderDate Order.PromiseDate
  VALIDATE(Order.PromiseDate > Order.OrderDate,
  "Promise date must be later than order date").
```

The previous statement is equivalent to the following statement:

```
SET Order.OrderDate Order.PromiseDate
  VALIDATE(INPUT Order.PromiseDate > Order.OrderDate,
  "Promise date must be later than order date").
```

The validation is based on the value of `OrderDate` prior to the SET statement. If you want to validate the value of `PromiseDate` against the input value of `OrderDate`, use this statement:

```
SET Order.OrderDate Order.PromiseDate
  VALIDATE(Order.PromiseDate > INPUT Order.OrderDate,
  "Promise date must be later than order date").
```

If you try to validate a field whose reference is ambiguous, the AVM tries to resolve the ambiguity by referencing the table that contains the record being updated. In the following example, the `SalesRep` field is ambiguous because it exists in both the Order table and the Customer table. The AVM resolves the ambiguity by validating the `SalesRep` field in the Order table, since the Order table is being updated.

```
FIND FIRST Customer.
FIND FIRST Order.
UPDATE Order.CustNum Order.SalesRep
  VALIDATE(LENGTH(Order.SalesRep) > 1, "Invalid sales rep value.").
```

If the reference is to an array field and has no subscript, the AVM assumes you want to use the subscript of the field that is being prompted.

If the value of *`condition`* is FALSE, use *`msg-expression`* to display a specific message. You must enclose *`msg-expression`* in quotation marks (" ").

The AVM processes validation criteria whenever the user attempts to leave the frame field. If the frame field value is not valid, the AVM displays *`msg-expression`* in the message area, causes the terminal to beep, and does not advance out of the frame field.

If you tab a frame field, make no changes, and leave the field, the AVM does not process the validation criteria specified with the VALIDATE option until the you press **GO** (**F1**). If you press **ENDKEY** or **END-ERROR**, or an error occurs, the AVM does not test the validation criteria specified with the VALIDATE option.

If the input source for the procedure is a table, the AVM validates each input field (except those with a value of "-"). If the result of the validation is FALSE, *`msg-expression`* is displayed and the AVM treats the validation as an error.

To suppress the Data Dictionary validation criteria for a field, use this VALIDATE option:

```
VALIDATE(TRUE,"")
```

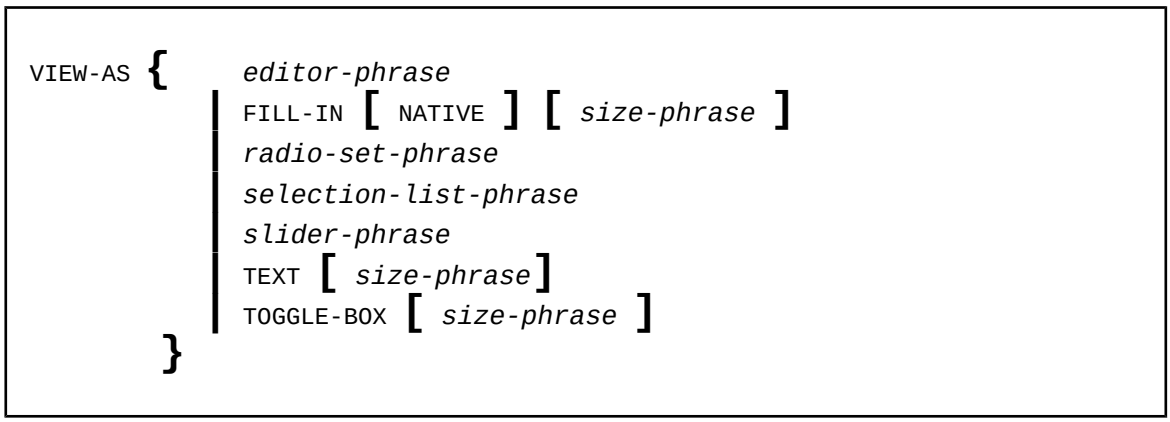
When you use the VALIDATE option in a procedure to specify validation criteria for a field, that validation criteria applies to all other references to that field in the same frame:

```
FOR EACH Order:
  UPDATE Order.OrderDate.
  UPDATE Order.OrderDate
    VALIDATE(Order.OrderDate LE TODAY, "Can't be later than today").
END.
```

In this example, the AVM applies the validation criteria on the second UPDATE statement. The AVM also applies the validation criteria to the first UPDATE statement because both UPDATE statements use the same frame. Scope references to the same field to different frames if you do not want a VALIDATE option to affect all references to that field.

view-as-phrase

Specifies the type of widget. This is the syntax for view-as-phrase:



For more information on *view-as-phrase*, see the VIEW-AS phrase reference entry.

WIDGET-ID *id-number*

Specifies a widget ID for a field or variable widget to display in a frame. The value of *id-number* must be an expression that evaluates to an even integer value between 2 and 65534, inclusive, and must be unique across all widget IDs in the window or dialog box.

If you specify an invalid ID, the compiler displays an error message. This option is supported in graphical interfaces only, and only in Windows.

Example

This procedure lets the user update Customer records after entering the password "secret." The format phrase on the Phone field describes the display format of that field.

r-frmat.p

```
DEFINE VARIABLE password AS CHARACTER NO-UNDO.

UPDATE password FORMAT "x(6)" BLANK
  VALIDATE(password = "secret", "Sorry, wrong password")
  HELP "Maybe the password is 'secret' !"
  WITH FRAME passw CENTERED SIDE-LABELS.
HIDE FRAME passw.

REPEAT:
  PROMPT-FOR Customer.CustNum COLON 20.  FIND Customer USING Customer.CustNum.
  UPDATE
    Customer.Name LABEL "Customer Name" COLON 20
      VALIDATE(Customer.Name NE "", "Please enter a name")
    Customer.Address HELP "Please enter two lines of address"
      COLON 20 LABEL "Address"
    Customer.Address2 NO-LABEL COLON 20
    Customer.City COLON 20
    Customer.State COLON 20
    Customer.PostalCode COLON 20 SKIP(3)
    Customer.Phone AT 5 FORMAT "(999) 999-9999"
    Customer.Contact TO 60
    WITH CENTERED SIDE-LABELS.
END.
```

Notes

- If you have enabled application-defined widget IDs in your ABL GUI application, by specifying the Use Widget ID (`-usewidgetid`) startup parameter, then the AVM uses the value specified in the WIDGET-ID option to set the WIDGET-ID attribute for this widget when it creates the widget at run time, instead of using the widget ID it normally generates by default. If you have not enabled application-defined widget IDs, then the AVM ignores this option setting at run time.

  For more information about the WIDGET-ID attribute, see its reference entry in the Handle Attributes and Methods Reference. For more information about the Use Widget ID (`-usewidgetid`) startup parameter, see *Startup Command and Parameter Reference*.

- For SpeedScript, these options are invalid: BGCOLOR, DCOLOR, FGCOLOR, FONT, PFCOLOR, *`view-as-phrase`*.

- With respect to internationalization, some double-byte and UTF-8 multi-byte characters display and print in one or two columns. Each unit in the format string represents one physical column. To display or print a character that requires two columns, the FORMAT phrase must specify two columns. For more information, see *Internationalize ABL Applications*.

See also

FORM statement, Frame phrase

Frame phrase

Specifies the overall layout and processing properties of a frame for frame definition (DEFINE FRAME and FORM), block header (DO, FOR EACH, and REPEAT), and data handling (DISPLAY, SET, etc.) statements. When used on block header statements, the Frame phrase also specifies the default frame for data handling statements within the block. Frame phrases can also be used on individual data handling statements to indicate the specific frame where the statement applies.

Syntax

```
WITH [ ACCUM [ max-length ] ]
     [ at-phrase ]
     [ CANCEL-BUTTON button-name ] [ CENTERED ]
     [ color-specification ]
     [ COLUMN expression ] [ n COLUMNS ]
     [ CONTEXT-HELP ] [ CONTEXT-HELP-FILE help-file-name ]
     [ DEFAULT-BUTTON button-name ]
     [ DROP-TARGET ]
     [ [ expression ] DOWN ] [ EXPORT ]
     [ WIDGET-ID id-number ] [ FONT expression ]
     [ FRAME frame ]
     [ INHERIT-BGCOLOR | NO-INHERIT-BGCOLOR ]
     [ INHERIT-FGCOLOR | NO-INHERIT-FGCOLOR ]
     [ KEEP-TAB-ORDER ] [ NO-BOX ]
     [ NO-HIDE ] [ NO-LABELS ] [ USE-DICT-EXPS ]
     [ NO-VALIDATE ] [ NO-AUTO-VALIDATE ]
     [ NO-HELP ] [ NO-UNDERLINE ]
     [ OVERLAY ] [ PAGE-BOTTOM | PAGE-TOP ] [ RETAIN n ]
     [ ROW expression ] [ SCREEN-IO | STREAM-IO ]
     [ SCROLL n ] [ SCROLLABLE ] [ SIDE-LABELS ]
     [ size-phrase ] [ STREAM stream | STREAM-HANDLE handle ] [ THREE-D ]
     [ title-phrase ] [ TOP-ONLY ] [ USE-TEXT ]
     [ V6FRAME [ USE-REVVIDEO | USE-UNDERLINE ] ]
     [ VIEW-AS DIALOG-BOX ] [ WIDTH n ] [ IN WINDOW window ]
```

ACCUM [*max-length*]

The ACCUM option lets you use aggregate functions (such as MAX, MIN, TOTAL, and SUBTOTAL) to accumulate values within shared frames. With the ACCUM option, aggregate values can be shared among procedures through shared frames. You must include the ACCUM option in the FORM statement or DEFINE FRAME statement of each procedure that uses the shared frame.

The following procedure defines a new shared frame:

```
DEFINE NEW SHARED FRAME x.

FORM field1 field2 WITH FRAME x ACCUM.
RUN testb.p.
```

The following procedure uses the shared frame:

```
/* testb.p */
DEFINE SHARED FRAME x.

FORM field1 field2 WITH FRAME x ACCUM.

FOR EACH table1:
  DISPLAY field1 field2 (TOTAL) WITH FRAME x.
END.
```

When you specify a user-defined aggregate label, use the *max-length* parameter of the ACCUM option to specify a maximum aggregate label length in the frame phrases of shared frames. For more information, see the Aggregate phrase reference entry.

at-phrase

Specifies the position of the frame (upper-left corner) within a window or parent frame. This is the syntax for the AT phrase for a frame:

```
AT { COLUMN column ROW row
   | X x Y y
   }
```

Note that for a frame parented by a window, you must specify an absolute position relative to the display area of the window. For a frame parented by another frame, you must specify a position relative to the display area of the parent frame. The default value for all AT phrase parameters is 1. ABL ignores the COLUMN or X option if you use the CENTERED option for the same frame. For more information on *at-phrase*, see the AT phrase reference entry.

CANCEL-BUTTON *button-name*

Specifies the cancel button for the frame. This is the button chosen when the **ESC** key code is applied to the frame in Windows. This button might also be chosen when the **ESC** key code is applied to a frame within the same frame family that does not have a cancel button. In such an event, the AVM searches the frame family in random order. The first cancel button found during this random search is chosen. The *button-name* argument must be a static button name.

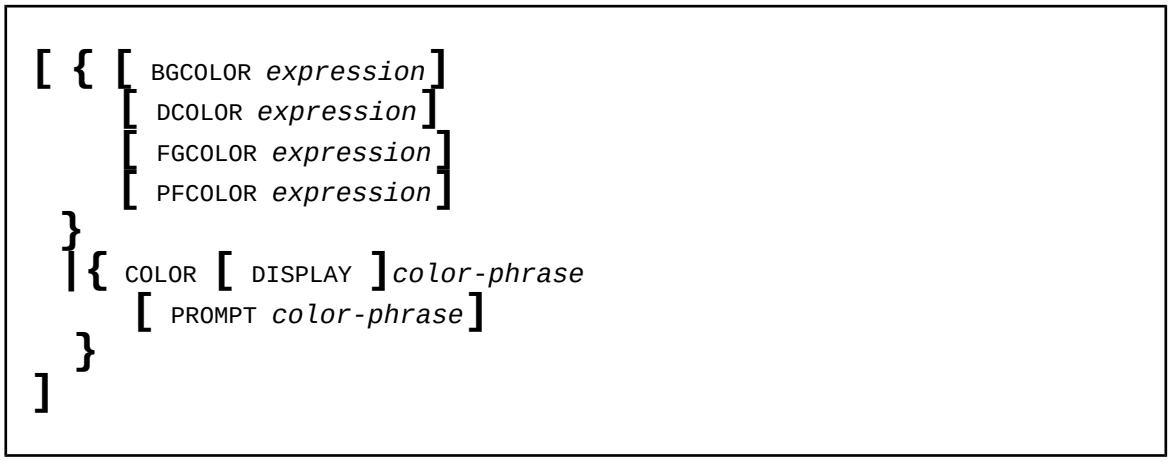
CENTERED

Centers the frame horizontally in the window or frame to which it is parented (or the terminal display, in character mode). If you use the CENTERED option and are sending output to a device other than the terminal, the AVM centers the frame for the terminal. This might result in a non-centered frame on the alternate output device.

You can also use the AT phrase or COLUMN option to specify the position of the frame.

color-specification

For a graphical user interface, specifies the foreground and background color of the frame; for a character interface, specifies the display and prompt colors for the frame.



For graphical interfaces, the FGCOLOR and BGCOLOR options specify the foreground and background color of the frame. These options are not supported in character interfaces. For character interfaces, use the DCOLOR and PFCOLOR options (which are not supported in graphical interfaces) to specify the display color and prompt color of the frame.

The COLOR option is obsolete, but is retained for backward compatibility.

Widgets (except child frames) within the frame inherit the colors of the frame by default. You can also set the colors of each widget individually.

COLUMN *expression*

The *expression* is a constant, field name, variable name or expression whose value is the number of the column, relative to the window or parent frame in which you place the frame. The default value is 1. ABL ignores this option if you use the CENTERED option for the same frame.

The AVM evaluates *expression* each time the frame comes into view or is printed at the top or bottom of a page (if the frame is a PAGE-TOP or PAGE-BOTTOM frame). For more information, see the *expression* option of the FORM statement.

n COLUMNS

Formats data fields into a specific number (*n*) of columns, where *n* can be 1, 2, or 3. For *n* = 1, 16 positions are allowed for a label; for *n* = 2, 14 positions are allowed; and for *n* = 3, 12 positions are allowed. Label positions include room for a colon and a space after the label. (If the -nocolon startup parameter was in effect when compiling the r-code, the colon is omitted, but room for the space after the label is still included. See the *Startup Command and Parameter Reference* for more information on -nocolon.) Labels are right justified if they are short, and truncated if they are too long. By default, the AVM wraps fields across the frame for as many lines as required, placing labels above the fields.

When you use this option, it forces SIDE-LABELS and overrides any AT, COLON, TO, or SPACE options you might have used in the same Frame phrase.

CONTEXT-HELP

Specifies that context-sensitive help is available for this frame. This option is valid in Windows GUI only.

CONTEXT-HELP-FILE *help-file-name*

Specifies the complete pathname of a help (.HLP) file associated with this frame. If CONTEXT-HELP-FILE is specified without CONTEXT-HELP, CONTEXT-HELP is assumed. This behavior can be overridden by setting the dialog box's CONTEXT-HELP attribute to FALSE at run time. This option is valid in Windows GUI only. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

DEFAULT-BUTTON *button-name*

Specifies a default button for the frame. This is the button chosen when the ENTER key code in Windows is invoked for the frame. This button might also be chosen when the **ESC** key code is applied to a frame within the same frame family that does not have a default button. In such an event, the AVM searches the frame family in random order. The first default button found during this random search is chosen. The *button-name* argument must be the name of a static button. This button must be defined with the DEFAULT option and cannot display an image.

DROP-TARGET

Indicates whether you want to be able to drop a file onto the object.

[*expression* **]** DOWN

Specifies that the frame is a down frame. A **down frame** is a frame that can display multiple occurrences of the set of fields defined in the frame. The *expression* is a constant, field name, variable name or expression whose value is the number of occurrences you want in the frame. If you specify 1 for *expression*, the frame is **not** a down frame.

Down frames are typically specified for iterative blocks. On the first iteration of the block, the AVM displays the first set of data (a record, field, or variable value) as the first occurrence in the frame. After displaying the data, the AVM advances to the next occurrence in the frame on the second iteration of the block, and displays the second set of data there. The AVM continues advancing and displaying data for the number of occurrences specified by *expression*, and prompts to continue with another set of occurrences until all the data has been displayed. The AVM evaluates *expression* each time the frame comes into view or is printed at the top or bottom of a page (if the frame is a PAGE-TOP or PAGE-BOTTOM frame). If you do not specify *expression*, the AVM displays as many occurrences as can fit in the current window.

If you do not use the DOWN option, ABL automatically makes certain frames down frames, unless you specify otherwise (1 DOWN).

EXPORT

This option is valid only for SQL.

WIDGET-ID *id-number*

Specifies a widget ID for a frame widget. The value of *id-number* must be an expression that evaluates to an even integer value between 2 and 65534, inclusive, and must be unique across all widget IDs in the window or dialog box.

If you specify an invalid ID, the compiler displays an error message. This option is supported in graphical interfaces only, and only in Windows.

FONT *expression*

Specifies the font of the frame. All widgets within a frame, except child frames, inherit the font of the frame by default. You can also set the font of each widget individually. By default, the AVM uses the default system font.

FRAME *frame*

Defines new frames by giving them unique names. Whenever the same frame name is referred to in more than one Frame phrase, ABL combines the characteristics on each Frame phrase naming that frame. ABL also combines any frame characteristics used in data handling statements that name the same frame into the same frame description. This option is redundant for DEFINE FRAME statements. If you do not specify this option, ABL uses the default frame for the current block.

INHERIT-BGCOLOR | NO-INHERIT-BGCOLOR

Specifies whether the following field-level widgets in the frame inherit the frame's background color: BROWSE, COMBO-BOX (all types), EDITOR, FILL-IN (NATIVE and Enabled), and SELECTION-LIST.

Specify the INHERIT-BGCOLOR option to allow field-level widgets in the frame to inherit the frame's background color. Specify the NO-INHERIT-BGCOLOR option to prevent field-level widgets in the frame from inheriting the frame's background color.

Specifying either of these options overrides the INHERIT-BGCOLOR attribute setting for the session. Setting the frame's INHERIT-BGCOLOR attribute overrides the INHERIT-BGCOLOR option on the frame. Setting the BGCOLOR attribute for a field-level widget within the frame overrides any INHERIT-BGCOLOR attribute or option settings.

INHERIT-FGCOLOR | NO-INHERIT-FGCOLOR

Specifies whether the following field-level widgets in the frame inherit the frame's foreground color: BROWSE, COMBO-BOX (all types), EDITOR, FILL-IN (NATIVE and Enabled), and SELECTION-LIST.

Specify the INHERIT-FGCOLOR option to allow field-level widgets in the frame to inherit the frame's foreground color. Specify the NO-INHERIT-FGCOLOR option to prevent field-level widgets in the frame from inheriting the frame's foreground color.

Specifying either of these options overrides the INHERIT-FGCOLOR attribute setting for the session. Setting the frame's INHERIT-FGCOLOR attribute overrides the INHERIT-FGCOLOR option on the frame. Setting the FGCOLOR attribute for a field-level widget within the frame overrides any INHERIT-FGCOLOR attribute or option settings.

KEEP-TAB-ORDER

Prevents the frame-oriented I/O statements, ENABLE, UPDATE, SET, and PROMPT-FOR, from changing the tab order of your widgets in the frame. The tab order always remains the same as the order in which you first specify widgets in the frame. If you do not specify this option, ABL creates a new tab order based on the order specified in each frame-oriented I/O statement.

All attributes and methods that affect tab order (such as FIRST-TAB-ITEM and MOVE-AFTER-TAB), continue to change the tab order whether or not you specify this option. If you specify the option, these attributes and methods specify a new tab order for all frame-oriented I/O statements that follow.

NO-BOX

Does not display a box around the frame. If you do not use this option, the AVM displays a box around the data you are displaying.

If you are sending data to a device other than a terminal and you do not use this option, the AVM omits the sides and bottom line of the box and replaces the top line with blanks.

`NO-HIDE`

Suppress the automatic hiding of the frame (when the block where the frame is scoped iterates). The frame is hidden only if space is needed to display other frames.

NO-HIDE suppresses hiding for a frame only when the block where that frame is scoped iterates. For example:

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.CustNum Customer.Name.
  FOR EACH Order OF Customer NO-LOCK:
    DISPLAY Order.OrderNum.
    DISPLAY "hello" WITH FRAME b COLUMN 60 NO-HIDE.
  END.
END.
```

In this example, the AVM does not hide frame b when the inner block iterates. However, it does hide frame b when the outer block iterates. If you want the frame to stay in view during iterations of the outer block, scope the frame to that block.

`NO-LABELS`

Does not display labels. This option overrides any COLUMN-LABEL option you include in another phrase or statement.

`NO-UNDERLINE`

Does not underline labels appearing above fields.

`USE-DICT-EXPS`

Ensures that validation expressions and help strings from the Data Dictionary are compiled into the application. Typically, when the ABL compiler encounters a field reference in an input statement, Data Dictionary help and validation expressions are compiled in for that field, unless the field has a HELP or VALIDATE option (format phrase) attached in the input statement (or earlier in the procedure). In this case, the custom help or validation expression is used.

There are two syntax constructs that can enable a field for input without the compiler specifically knowing about it: ENABLE ALL and *`widget-name`*:SENSITIVE = YES.

When ABL encounters an ENABLE ALL statement, every field in the associated frame has Data Dictionary validation expressions and help strings compiled into the application. This closes any possible validation or help hole. As a side-effect, validation expressions and help strings that are not required might be compiled, but this will not affect the application.

This behavior places two important conditions on you. First, adding a field to a frame after the first ENABLE ALL is not desirable. Data Dictionary validation and help will not be compiled for this field. Second, any custom validation or help must come before the first ENABLE ALL. A good practice is to include these in the DEFINE FRAME or FORM statements.

In the case of *widget-name*:SENSITIVE = YES, there is more potential for validation and help holes. Since the compiler cannot predict whether these statements are used, in effect, as input statements, no help or validation is compiled. USE-DICT-EXPS explicitly compiles in all validation expressions and help strings for a frame. For each frame that you use *widget-name*:SENSITIVE = YES, specify USE-DICT-EXPS. This closes any potential validation or help holes. To provide custom help or validation when using USE-DICT-EXPS, the HELP or VALIDATE option must appear in the first reference to that field. Typically, this is in the DEFINE FRAME or FORM statement.

NO-VALIDATE

Disregards all validation conditions specified in the Data Dictionary for fields entered in this frame.

NO-AUTO-VALIDATE

Tells ABL to compile into the code all relevant validations it finds in the OpenEdge Data Dictionary, but to run the validations only when the code for the frame or for a field-level child-widget of the frame specifically invokes the VALIDATE() method.

NO-HELP

Disregards all help strings specified in the Data Dictionary for fields entered in this frame.

OVERLAY

Indicates that the frame can overlay any other frame that does not use the TOP-ONLY option. If you do not use this option, the frame you are using cannot overlay other frames. If the AVM needs to display an OVERLAY frame and doing so will partially obscure a TOP-ONLY frame, it first hides the TOP-ONLY frame. Any frame parented by another frame is an OVERLAY frame within the parent frame.

This procedure uses the OVERLAY option on the Frame phrase:

r-ovrlay.p

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer WITH 2 COLUMNS
    TITLE "Customer Information".
  FOR EACH Order OF Customer NO-LOCK:
    DISPLAY Order WITH 2 COLUMNS OVERLAY
      TITLE "Customer's Orders" ROW 7 COLUMN 10.
  END.
END.
```

The procedure above displays customer information in one frame. The procedure then displays order information for the customer in a second frame that overlays the first.

PAGE-BOTTOM

Displays the frame at the bottom of the page each time the output ends a page.

PAGE-TOP

Displays the frame each time the output begins on a new page.

The following table shows how the PAGE-TOP and PAGE-BOTTOM options work depending on the kind of DISPLAY or VIEW.

Table 42: Using PAGE-TOP and PAGE-BOTTOM frames

Pseudo-code	OUTPUT to a PAGED file	OUTPUT to the screen or an UNPAGED file
DISPLAY a PAGE-TOP Frame	The frame is written to the file and put on a list of frames to be written at the top of each page.	The frame is written to the file or displayed on the screen.
VIEW a PAGE-TOP Frame	The frame is put on a list of frames to be written at the top of each page.	The frame is written to the file or displayed on the screen.
DISPLAY a PAGE-BOTTOM Frame	The frame is written to the file and put on a list of frames to be written at the bottom of each page.	The frame is written to the file or displayed on the screen.
VIEW a PAGE-BOTTOM Frame	The frame is put on a list of frames to be written at the bottom of each page.	The frame is written to the file or displayed on the screen.
HIDE Either Type	The frame is removed from the appropriate list.	The frame is removed from the screen.

`RETAIN` *`n`*

Specifies the number of frame iterations to retain when the frame scrolls on the screen. The *`n`* must be a constant. For example, RETAIN 2 causes the AVM to display the last two iterations in a down frame at the top of the frame. If you are using UP to scroll up a window, those two lines are displayed at the bottom of the window. Do not use the SCROLL option in a Frame phrase in which you also use the RETAIN option. By default, the AVM does not retain any iterations in the window that have already been displayed.

`ROW` *`expression`*

The *`expression`* is a constant, field name, variable name, function reference, or expression whose value is the row, relative to the window or parent frame in which you place the frame. If you are displaying a frame on a device other than a terminal, this option has no effect. By default, the AVM displays a root frame at the next available row of the window and displays a child frame at row 1 of the parent frame.

The AVM evaluates *`expression`* each time the frame comes into view or is printed at the top or bottom of a page (if the frame is a PAGE-TOP or PAGE-BOTTOM frame).

For more info, see the *`expression`* option of the FORM statement.

[`SCREEN-IO` **|** `STREAM-IO` **]**

If you specify STREAM-IO for a frame, the USE-TEXT option is assumed and all font specifications are ignored. The frame is formatted using a fixed font in a manner appropriate for streaming to a text file or printer. In particular, all border padding for FILL-IN widgets is dropped and the default system font is used.

If you use the STREAM-IO option on the COMPILE statement, this behavior is the default for all frames in the procedure. In this case, you can override that option by specifying SCREEN-IO for an individual frame.

SCROLL *n*

Displays a scrolling frame rather than a paging frame. The value *n* is a constant that specifies the number of frame iterations to scroll when the frame scrolls in the window. For example, if a procedure uses a DISPLAY or DOWN statement when a scrolling frame is full, the data in the frame scrolls up *n* iterations (rather than clearing and repainting the frame as it would without the SCROLL option).

This procedure uses the SCROLL option to scroll the display one line at a time:

r-fphrsc.p

```
FOR EACH Customer NO-LOCK WHERE Customer.CustNum <= 50:
  DISPLAY Customer.CustNum Customer.Name Customer.CreditLimit
    WITH SCROLL 1 USE-TEXT.
  IF Customer.CreditLimit >= 50000 THEN
    COLOR DISPLAY MESSAGES Customer.CreditLimit.
END.
```

Do not use the RETAIN option in a Frame phrase in which you also use the SCROLL option.

SCROLLABLE

If you specify this option, the virtual size of the frame might exceed the physical space allocated for it in the window. If that happens, scrolling is enabled for the frame. If you omit this option, the physical and virtual size of the frame are always the same and scrolling is never enabled for the frame.

SIDE-LABELS

Displays field labels to the left of and centered against the data, separated from the data by a colon (:) and a space. If you do not use the SIDE-LABELS option, the AVM displays labels above their corresponding fields in the frame header and separates the labels from the field values with underlining. To create side labels without colons automatically added, compile the r-code with the `-nocolon` startup parameter in effect. See the *Startup Command and Parameter Reference* for more information.

size-phrase

Specifies the size of the frame. This is the syntax for *size-phrase*:

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

For more information on *size-phrase*, see the SIZE phrase reference entry.

STREAM *stream*

Allows you to specify the name of a stream for SQL statements.

STREAM-HANDLE *handle*

Specifies the handle to a stream. If *handle* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

THREE-D

Specifies that the frame and all contained widget appear in three-dimensional format (Windows only). If you specify the THREE-D option for a frame, the default background color is gray rather than the window color. Frames do not inherit the THREE-D setting from a parent window, and child frames do not inherit the THREE-D setting from a parent frame.

title-phrase

Displays a title as part of the top line of the box around a display frame. Following is the syntax for the *title-phrase*:

```
TITLE
  [ { [ BGCOLOR expression ]
      [ DCOLOR expression ]
      [ FGCOLOR expression ]
    }
  | COLOR color-phrase ]
  [ FONT expression ] title-string
```

The *title-string* is a constant, field name, variable name, or expression whose result is a character value. The *expression* is the value you want to display as a title. If *title-string* is a constant character string, it must be surrounded by quotes (""). The AVM automatically centers *title-string* in the top line of the frame box.

The BGCOLOR, FGCOLOR, COLOR, and FONT options have no effect and are supported for backward compatibility only.

You can use the DCOLOR option to specify the color of the title in a character interface.

TOP-ONLY

Indicates that no other frame can overlay this frame. If you do not use this option, other frames that use the OVERLAY option can overlay this frame. If the AVM has to display an OVERLAY frame and by doing so will partially obscure a TOP-ONLY frame, it first hides the TOP-ONLY frame. See also the OVERLAY attribute reference entry.

USE-TEXT

Specifies that the default widget type for all widgets in the frame is TEXT rather than FILL-IN. Thus, all border padding on the widgets is dropped.

V6FRAME [USE-REVVIDEO | USE-UNDERLINE]

The V6FRAME option is designed specifically to compile and run Progress Version 6 applications with Progress Version 7 or later in Windows. This option uses the V6FontNumber setting in the [Startup] section of the current environment to calculate the height and width of a character unit and then set the layout grid used to compile frames for display in Progress Version 7 or later.

At run time, the FONT attribute for a frame compiled with the V6FRAME option is set to the font number specified with the V6FontNumber setting. The default setting for the V6FontNumber setting is 3.

By default, V6FRAME displays a border around a fill-in field. This means that your code requires more space on the screen than in Progress Version 6. You can override this behavior with one of the following options:

- USE-REVVIDEO displays no border around a fill-in field. When a fill-in is enabled for input, the color of the fill-in changes to the color specified with the INPUT setting in the [Colors] section in the current environment. The IBEAM cursor signals that a fill-in field has input focus.
- USE-UNDERLINE displays no border around a fill-in widget. When a fill-in is enabled for input, the underline attribute of the font (V6FontNumber) for the fill-in is turned on. The color of a fill-in enabled for input does not change. The IBEAM cursor signals that a fill-in field has input focus.

The V6FRAME option also limits the vertical size of a frame title to one character unit based upon the layout grid. The text of the frame title is in the font specified with the V6FontNumber setting in the [Startup] section of the current environment.

The V6FRAME option governs the appearance of screen output only. Use the STREAM-IO option to compile procedures that output to files and printers. If you specify the V6FRAME and STREAM-IO options in the same frame phrase, the STREAM-IO option overrides the V6FRAME option.

For more information on the environment for an ABL session, see *Manage ABL Applications*.

`VIEW-AS DIALOG-BOX`

Specifies that the frame is displayed as a dialog box. A dialog box is a modal, one-down frame with many of the properties of a window. Like a window, a dialog box can be moved and programmatically resized, and it acquires scroll bars when it is resized smaller than its original frame dimensions. Unlike a window, it cannot be minimized or maximized; nor can it have a menu bar. As a frame-level widget, it is owned by a window and can contain a frame family, but it cannot be owned by another frame or dialog box. Because it is modal, a dialog box must be disabled before any other widgets in the application can be accessed by the user. For more information on the properties of a dialog box, and to compare them with the properties of a frame, see the Widget Reference.

`WIDTH` *`n`*

Specifies the number (*n*) of columns in a frame. If you do not use *`size-phrase`* or the WIDTH option, the width of the frame is based on the fields you are displaying, the position of the frame, and the width of the current or specified window.

`IN WINDOW` *`window`*

Specifies the window in which the frame is displayed. The value *`window`* must be the handle of a window. This option is not allowed in a DISABLE statement. By default, the AVM displays the frame in the current window.

Examples

The `r-frame.p` procedure displays the `CustNum`, `Name`, and `Phone` number for each Customer record. The frame phrase (starting with the word WITH) describes the frame being used to display that information.

r-frame.p

```
FOR EACH Customer NO-LOCK:
  FORM HEADER
    "No-box, No-Underline, No-labels, 5 DOWN" SKIP
    "Centered" SKIP(2)
    WITH NO-BOX NO-UNDERLINE NO-LABELS CENTERED 5 DOWN.
  DISPLAY Customer.CustNum Customer.Name Customer.Phone.
END.
```

The r-frame2.p procedure produces a Customer report, using Customer List as the header for each page of the report and using Customer List Continued On Next Page as the footer for each page of the report. The OUTPUT TO statement directs all output to the file phone.lst. After running the r-frame2.p procedure, you can press **GET** then type the name of the file to view the contents of phone.lst.

r-frame2.p

```
OUTPUT TO phone.lst PAGED PAGE-SIZE 20.
FOR EACH Customer NO-LOCK:
  FORM HEADER "Customer List" AT 1 PAGE-NUMBER TO 60
    WITH FRAME hdr PAGE-TOP CENTERED NO-BOX.
  VIEW FRAME hdr.
  FORM "Customer List Continued On Next Page"
    WITH FRAME footr PAGE-BOTTOM CENTERED.
  VIEW FRAME footr.

  DISPLAY Customer.CustNum Customer.Name Customer.Phone WITH CENTERED.
END.
HIDE FRAME footr.
OUTPUT CLOSE.
```

Notes

- PAGE-TOP and PAGE-BOTTOM frames are activated based on DISPLAY or VIEW statements as previously described. They are deactivated when the block in which the frames are scoped iterates or ends.
- If you use the SIZE phrase for a down frame, then the size you specify determines the number of iterations in the frame. The number of iterations you specify with the DOWN option is ignored.
- You can input and output to a frame only when that frame is in full view. Therefore, when you input or output to a frame that is hidden or partially overlayed, the AVM displays the frame first.
- An empty WITH clause is valid. If the WITH keyword appears by itself, or in the clause following an earlier WITH, it is ignored. This feature is useful when designing template programs to be called with arguments. For example, a template program with a line like DISPLAY {1} WITH {2} executes correctly even if called with only one argument.
- The SIZE phrase and WIDTH options are mutually exclusive. If you specify WIDTH or you specify neither WIDTH nor the SIZE, the height of a frame is based on the fields you are displaying, the position of the frame, and whether or not it is a down frame.
- A frame parented by another frame cannot function as a down frame.
- If you position a child frame completely outside the virtual area of its parent frame, the AVM raises ERROR at run time when the frame is realized.
- If you position a child frame partially within the virtual area of its parent frame or the child frame is larger than the virtual area of the parent frame, the AVM crops the child frame to fit the parent's virtual area and adds scroll bars to the child.
- If you position a child frame partially within the physical area of its parent frame or the child frame is larger than the physical area of the parent frame, the AVM adds scroll bars to the parent.
- You cannot specify the VIEW-AS DIALOG-BOX option for a frame used as a DDE frame. For information on DDE frames, see *OpenEdge Programming Interfaces*.
- If you have enabled application-defined widget IDs in your ABL GUI application, by specifying the Use Widget ID (-usewidgetid) startup parameter, then the AVM uses the value specified in the WIDGET-ID option to set the WIDGET-ID attribute for this widget when it creates the widget at run time, instead of using the widget ID it normally generates by default. If you have not enabled application-defined widget IDs, then the AVM ignores this option setting at run time.

For more information about the WIDGET-ID attribute, see its reference entry in the Handle Attributes and Methods Reference. For more information about the Use Widget ID (`-usewidgetid`) startup parameter, see *Startup Command and Parameter Reference*.

- For SpeedScript, WebSpeed evaluates the Frame phrase as though you were running a character client. The typical WebSpeed application does not use frames when defining layout. However, if you are using existing ABL code that includes frame layouts, you can iterate through frame children to retrieve validation expressions and help strings. Generally, in SpeedScript programming, the frame serves as a virtual container for widgets. These options are invalid: CENTERED, CONTEXT-HELP, CONTEXT-HELP-FILE, DEFAULT-BUTTON, SCROLLBAR VERTICAL, V6FRAME, USE-REVVIDEO, USE-UNDERLINE, VIEW-AS DIALOG-BOX, IN WINDOW.

See also

DEFINE FRAME statement, FORM statement, Format phrase, FRAME-COL function, FRAME-DOWN function, FRAME-LINE function, FRAME-ROW function, Stream object handle

FRAME-COL function

Returns a DECIMAL value that is the column position of the left corner of a frame within its window.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-COL [ ( frame ) ]
```

frame

The name of the frame whose column position you are trying to determine. If you do not supply a frame name, the FRAME-COL function uses the default frame for the block it is in. If the FRAME-COL function is in a DO block, the function uses the default frame scoped to the block containing the DO block.

Example

This procedure displays Customer information in one frame, then displays Order information in an overlay frame. FRAME-ROW places the overlay frame on the ninth row of the second column. FRAME-COL places the overlay frame on the first column of the first frame.

r-frcol.p

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer WITH FRAME cust-frame 2 COLUMNS
    TITLE "CUSTOMER INFORMATION".
  FOR EACH Order OF Customer NO-LOCK:
    DISPLAY Order.OrderNum Order.OrderDate Order.ShipDate Order.PromiseDate
      Order.Carrier Order.Instructions Order.PO
      WITH 2 COLUMNS 1 DOWN OVERLAY TITLE "CUSTOMER'S ORDERS"
        ROW FRAME-ROW(cust-frame) + 8 COLUMN FRAME-COL(cust-frame) + 1.
  END.
END.
```

Notes

- The FRAME-COL function returns a value of 0 if the frame you specify is not in view when the AVM evaluates the function.
- To convert the decimal value returned by FRAME-COL to an integer value, use the INTEGER function.

See also

Frame phrase, FRAME-DOWN function, FRAME-LINE function, FRAME-ROW function, INTEGER function

FRAME-DB function

Returns the logical database name of the database that contains any field in which the user-interface cursor is entered.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-DB
```

The function requires no arguments. If the cursor is in a field that is not a database field, this function returns no value for the field.

Example

For each field being updated, this procedure displays the field name, the table the field belongs to, and the database in which the table exists. The EDITING phrase is part of the UPDATE statement; it displays information on the field as you update the record, and then reads each of the keystrokes entered (READKEY) and applies those keystrokes (APPLY LASTKEY).

r-frdb.p

```
FOR EACH Customer NO-LOCK:
  UPDATE Customer.CustNum Customer.Name Customer.Address Customer.Address2
    Customer.City Customer.State Customer.PostalCode
    WITH 1 DOWN 1 COLUMN CENTERED EDITING:
    DISPLAY "You are editing field: " FRAME-FIELD SKIP
      " of file: " FRAME-FILE SKIP
      " in database: " FRAME-DB
      WITH FRAME a ROW 15 NO-LABELS CENTERED.
    READKEY.
    APPLY LASTKEY.
  END. /* EDITING */
END.
```

Notes

- If the cursor is not in an enabled input field when the last input statement is executed, or the input field is not associated with a database field, FRAME-DB returns an empty string.

- Use this syntax to find the name of a schema holder for a non-OpenEdge database:

```
SDBNAME ( FRAME-DB )
```

See also

DBCODEPAGE function, DBCOLLATION function, FRAME-FIELD function, FRAME-FILE function, FRAME-INDEX function, LDBNAME function, PROGRAM-NAME function, SDBNAME function

FRAME-DOWN function

Returns an INTEGER value that represents the number of iterations in a frame.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-DOWN [ ( frame ) ]
```

frame

The name of the frame whose number down you are trying to determine. If you do not supply a frame name, the FRAME-DOWN function uses the default frame for the block it is in. If the FRAME-DOWN function is in a DO block, the function uses the default frame scoped to the block containing the DO block.

Example

This procedure displays Customers in a frame. When the frame is full, the procedure prompts "Do you want to see the next page?" The procedure recognizes that the frame is full when the value of FRAME-LINE (current logical line number) equals the value of FRAME-DOWN (number of iterations in the frame).

r-frdown.p

```
DEFINE VARIABLE ans AS LOGICAL NO-UNDO.

REPEAT:
  FIND NEXT Customer NO-LOCK.
  DISPLAY Customer.CustNum Customer.Name.
  IF FRAME-LINE = FRAME-DOWN THEN DO:
    MESSAGE "Do you want to see the next page ?" UPDATE ans.
    IF NOT ans THEN LEAVE.
  END.
END.
```

Note

The FRAME-DOWN function returns a value of 0 if used with a single frame or if the frame is not in view when the function is evaluated.

See also

Frame phrase, FRAME-COL function, FRAME-LINE function, FRAME-ROW function

FRAME-FIELD function

During a data entry statement, returns the name of the input field the cursor is in. At other times, returns the name of the input field the cursor was last in.

The FRAME-FIELD function is particularly useful if you want to provide the user with help for the input field being used.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-FIELD
```

Example

For each field the user is updating, this procedure displays the name of the field, the table the field belongs to, and the value currently in the field. The EDITING phrase is part of the UPDATE statement; it displays information on the field as the user updates the record, and then reads each of the keystrokes entered (READKEY) and applies those keystrokes (APPLY LASTKEY).

r-frfld.p

```
FOR EACH Customer:
  UPDATE Customer.CustNum Customer.Name Customer.Address Customer.Address2
    Customer.City Customer.State Customer.PostalCode
    WITH 1 DOWN 1 COLUMN CENTERED EDITING:
    DISPLAY "You are editing field:" FRAME-FIELD SKIP
      "of file:" FRAME-FILE SKIP
      "Its value is:" FRAME-VALUE FORMAT "x(20)"
      WITH FRAME a ROW 15 NO-LABELS CENTERED.
    READKEY.
    APPLY LASTKEY.
  END. /* EDITING */
END.
```

Notes

- If the current or last input field is an array, FRAME-FIELD returns the name of the field but does not indicate the array element that the input field represents. To display the array element, use the FRAME-INDEX function.
- If the cursor was not in an enabled input field when the last input statement ended, FRAME-FIELD returns an empty string.

See also

FRAME-FILE function, FRAME-INDEX function, FRAME-VALUE function, PROGRAM-NAME function

FRAME-FILE function

Returns the name of the database table that contains the field the cursor is in. The FRAME-FILE function is useful if you want to provide users with context-sensitive help.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-FILE
```

Example

This procedure updates fields from the Order table and the Customer table. It uses the FRAME-FILE function to tell you which table contains the field being updated.

r-frfile.p

```
FOR EACH Customer, EACH Order OF Customer:
  DISPLAY Order.OrderNum WITH CENTERED ROW 2 FRAME onum.
  UPDATE
    Customer.CustNnum AT 5 Order.CustNum AT 30 SKIP
    Customer.Name AT 5
    Customer.City AT 5
    Customer.State AT 5
    Customer.PostalCode AT 5
    WITH ROW 8 CENTERED 1 DOWN NO-LABELS EDITING:
    MESSAGE "The field" FRAME-FIELD "is from the" FRAME-FILE "file".
    READKEY.
    APPLY LASTKEY.
  END. /* EDITING */
END.
```

Notes

- FRAME-FILE returns a null string if the frame field being entered is not associated with a database field.
- If the cursor is not in an enabled input field when the last input statement ends, FRAME-FILE returns a null string.
- The FRAME-FILE value is set to blanks at the next PAUSE statement, at the next READKEY statement, or when the AVM pauses automatically.

See also

FRAME-FIELD function, FRAME-VALUE function, PROGRAM-NAME function

FRAME-INDEX function

During a data entry statement, returns the subscript of the array element of the input field that the cursor is in as an INTEGER value. At other times, returns the subscript of the array element the cursor was in.

The FRAME-INDEX function is particularly useful if you want to provide the user with help for the input array element being edited.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-INDEX
```

Example

In this example, the FRAME-INDEX function uses the cursor position to determine which option you have chosen:

r-frindx.p

```
DEFINE VARIABLE menu AS CHARACTER NO-UNDO EXTENT 3.

DO WHILE TRUE:
  DISPLAY
    "1. Display Customer Data" @ menu[1] SKIP
    "2. Display Order Data"    @ menu[2] SKIP
    "3. Exit"                  @ menu[3] SKIP
    WITH FRAME choices NO-LABELS.
  CHOOSE FIELD menu AUTO-RETURN WITH FRAME choices
    TITLE "Demonstration Menu" WITH CENTERED ROW 10.
  HIDE FRAME choices.
  IF FRAME-INDEX EQ 1 THEN
    MESSAGE "You picked option 1.".
  ELSE IF FRAME-INDEX EQ 2 THEN
    MESSAGE "You picked option 2.".
  ELSE IF FRAME-INDEX EQ 3 THEN LEAVE.
END.
```

Notes

- If the cursor is not in an enabled input field when the last input statement is executed, FRAME-INDEX returns a 0. For example, FRAME-INDEX returns 0 if the user presses **END-ERROR** on the previous input statement.
- The FRAME-INDEX value is set to 0 at the next pause (done by a PAUSE statement or automatically by the AVM) or at the next READKEY statement.
- If you are updating a subset of an array-for example, `TEXT(array-field[i TO 12])`, and you use a variable subscript `(i)`, then FRAME-INDEX returns 0. If you use a constant subscript, `TEXT(array-field[6 TO 12])`, then FRAME-INDEX returns the correct value.

See also

Frame phrase, FRAME-DB function, FRAME-FIELD function, FRAME-FILE function

FRAME-LINE function

Returns an INTEGER value that represents the current logical line number in a down frame.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-LINE [ ( frame ) ]
```

frame

The frame name that you are trying to determine a line number for. If you do not supply a frame name, the FRAME-LINE function uses the default frame for the block that contains the FRAME-LINE function. If the FRAME-LINE function is in a DO block, the function uses the default frame scoped to the block that contains the DO block.

Example

This procedure lists Customers and allows the user to delete Customers one at a time. When the user presses **GET** to delete a Customer, the procedure displays an overlay frame below the last Customer displayed. The overlay frame prompts "Do you want to delete this customer?" The user answers yes or no. The AVM calculates the position of the overlay frame from the upper-right corner of the frame and the current line within the frame. That is, FRAME-ROW + 3 + FRAME-LINE gives the position of the current line in the frame, taking into account the three lines for the frame box and the labels. The prompt is placed five lines below the current line.

r-frline.p

```
DEFINE VARIABLE ans AS LOGICAL NO-UNDO
  LABEL "Do you want to delete this customer ?".

IF KBLABEL("GET") = "GET" THEN ON F3 GET.

STATUS INPUT "Enter data, or use the " + KBLABEL("get")
  + " key to delete the customer".

get-cust:
FOR EACH Customer WITH 10 DOWN:
  UPDATE Customer.CustNum Customer.Name Customer.CreditLimit EDITING:
    READKEY.
    IF KEYFUNCTION(lastkey) = "get" THEN DO:
      UPDATE ans WITH ROW FRAME-ROW + 3 + FRAME-LINE + 5
        COLUMN 10 SIDE-LABELS OVERLAY FRAME del-frame.
      IF ans THEN DO:
        DELETE Customer.
        NEXT get-cust.
      END.
    END.
    APPLY LASTKEY.
  END.
END.
```

Notes

- If there is a down pending for a frame, the FRAME-LINE function returns a value equal to FRAME-LINE + 1.
- The FRAME-LINE function counts an underline row as a logical line. A logical line corresponds to one iteration in a down frame and can contain more than one physical line.
- The FRAME-LINE function returns a value of 0 if the frame is not in view when the function is evaluated.

See also

Frame phrase, FRAME-COL function, FRAME-DOWN function, FRAME-ROW function

FRAME-NAME function

Returns the name of the frame that the cursor is in to a field that is enabled for input.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-NAME
```

Example

This procedure displays Customer information in one frame, then displays Order information for the Customer in a second frame. Use the FRAME-NAME function to display the name of the frame the cursor is in.

r-frname.p

```
FOR EACH Customer, EACH Order OF Customer:
  DISPLAY Order.OrderNum WITH CENTERED ROW 2 FRAME onum.
  UPDATE Customer.CustNum AT 5 Customer.Name AT 30 SKIP
    WITH FRAME custfrm WITH CENTERED 1 DOWN EDITING:
    DISPLAY " You are currently editing a frame called " FRAME-NAME
      WITH FRAME d1 WITH 1 DOWN CENTERED.
    READKEY.
    APPLY LASTKEY.
    IF LASTKEY = KEYCODE("RETURN") THEN
      MESSAGE " Press the space bar to edit order shipdate".
  END. /* Editing */
  HIDE FRAME custfrm.
  HIDE FRAME d1.
  UPDATE Order.ShipDate AT 5
    WITH FRAME orderfrm WITH CENTERED 1 DOWN EDITING:
    DISPLAY " Now you are editing a frame called" FRAME-NAME
      WITH FRAME d2 WITH 1 DOWN CENTERED.
    READKEY.
    APPLY LASTKEY.
  END.
  HIDE FRAME orderfrm.
  HIDE FRAME d2.
END.
```

Notes

- The FRAME-NAME function returns an empty string for a frame that has not been named (the default frame). It also returns an empty string if the cursor is in a field that is not enabled for input.
- When using the FRAME-NAME function, you must place it logically following the Frame phrase where it is named.
- FRAME-NAME is especially useful for context-sensitive help.

See also

Frame phrase, PROGRAM-NAME function

FRAME-ROW function

Returns a DECIMAL value that represents the row position of the upper-left corner of a frame within its window.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-ROW [ ( frame ) ]
```

frame

The name of the frame whose row position you are trying to determine. If you do not supply a frame name, the FRAME-ROW function uses the default frame for the block that contains the FRAME-ROW function. If the FRAME-ROW function is in a DO block, the function uses the default frame scoped to the block that contains the DO block.

Example

This procedure displays Customer information in one frame, then displays Order information for the Customer in a second frame that overlays the first. FRAME-ROW and FRAME-COL control the placement of the overlay frame. FRAME-ROW places the overlay frame on the eighth row of the first frame. FRAME-COL places the overlay frame on the first column of the first frame.

r-frrow.p

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer WITH FRAME cust-frame 2 COLUMNS
    TITLE "CUSTOMER INFORMATION".
  FOR EACH Order OF Customer NO-LOCK:
    DISPLAY Order.OrderNum Order.OrderDate Order.ShipDate
      Order.PromiseDate Order.Carrier Order.Instruction Order.PO
      WITH 2 COLUMNS 1 DOWN OVERLAY TITLE "CUSTOMER'S ORDERS"
      ROW FRAME-ROW(cust-frame) + 8 COLUMN FRAME-COL(cust-frame) + 1.
  END.
END.
```

Note

To convert the decimal value returned by FRAME-ROW to an integer value, use the INTEGER function.

See also

Frame phrase, FRAME-COL function, FRAME-DOWN function, FRAME-LINE function, INTEGER function

FRAME-VALUE function

During a data entry statement, returns the (character string) value of the input field that the cursor is in to the current input field. At other times, returns the (character string) value of the input field the cursor was last in.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-VALUE
```

Example

When the user presses **END-ERROR** while running this procedure, the procedure displays the name and value of the field the user was updating, along with the name of the table that contains that field.

r-frval.p

```
FOR EACH Customer:
  DISPLAY Customer.CustNum Customer.Name Customer.Address Customer.City
    Customer.State Customer.PostalCode
    WITH 1 COLUMN DOWN COLUMN 20.
  SET Customer.Address Customer.City Customer.State Customer.PostalCode.
END.

DISPLAY
  "You were updating field:" FRAME-FIELD FORMAT "x(20)" SKIP
  "Of file:" FRAME-FILE SKIP
  "And it had the value:" FRAME-VALUE FORMAT "x(20)" SKIP(2)
  "When you pressed the END-ERROR key."
  WITH FRAME helper NO-LABELS COLUMN 20 ROW 14 NO-BOX.
```

Notes

- If the cursor is not in an enabled input field when the last input statement ends, FRAME-VALUE returns a null string.
- FRAME-VALUE is set to blanks at the next pause (done by a PAUSE statement or automatically by the AVM) or at the next READKEY statement.
- FRAME-VALUE returns strings. If you use FRAME-VALUE to return a number, you must convert it prior to numeric comparisons. For example:

```
FIND Customer WHERE Customer.CustNum = INTEGER(FRAME-VALUE)
```

See also

FRAME-FIELD function, FRAME-FILE function, FRAME-VALUE statement, PROGRAM-NAME function

FRAME-VALUE statement

Stores the value of an expression in a frame field during a data entry statement.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-VALUE = expression
```

expression

A constant, field name, variable name or expression whose value you want to store in a frame field. If no frame is active when the AVM runs this statement, the AVM returns an error message. Otherwise, if the frame is in view, the AVM redisplays the field.

The data type of the *expression* must be the same as the data type of the frame field in which you are storing that expression. However, if the data type of *expression* is character, the AVM stores characters in the frame field regardless of the data type of that frame field, truncating characters if necessary.

The FRAME-VALUE statement can pass information from an `applhelp.p` procedure to the calling procedure. For example, if the user enters a value into a field called help-field, you can pass that value back to the calling procedure with this statement:

```
FRAME-VALUE = INPUT help-field.
```

Example

This procedure displays the word PROGRESS, the date, and a message instructing you to enter data or press the **GET** key to enter the Unknown value (?). You can update the information in the frame. If you press GET, the `r-frmval.p` procedure assigns the Unknown value (?) to a field with the FRAME-VALUE statement.

r-frmval.p

```
DEFINE VARIABLE txt     AS CHARACTER NO-UNDO INITIAL "PROGRESS".
DEFINE VARIABLE tmpdate AS DATE      NO-UNDO INITIAL TODAY.

IF KBLABEL("GET") = "GET" THEN ON F3 GET.

STATUS INPUT "Enter data or use the " + KBLABEL("GET")
  + " key to enter the Unknown value (?)".

UPDATE txt tmpdate EDITING:
  READKEY.
  IF KEYFUNCTION(LASTKEY) = "GET" THEN DO:
    FRAME-VALUE = ?.
    NEXT.
  END.
  APPLY LASTKEY.
END.
```

See also

FRAME-FIELD function, FRAME-FILE function, FRAME-VALUE function

FUNCTION statement

Defines or declares a prototype for a user-defined function, or declares a Web service operation. The following syntax boxes describe the syntax for each use of the statement, beginning with a user-defined function definition.

Syntax

```
FUNCTION function-name [ RETURNS ] return-type [ PRIVATE ]
  [ ( parameter[ , parameter ] ... ) ] :

  function-body
```

Use the following syntax to declare a user-defined function prototype that is defined later in the same procedure or that is defined in another external procedure:

```
FUNCTION function-name [ RETURNS ] return-type
  [ ( parameter[ , parameter ] ... ) ]
  {    FORWARD
     | [ MAP [ TO ] actual-name ] IN proc-handle
     | IN SUPER
  } .
```

Use the following syntax to declare a Web service operation.

```
FUNCTION operationName [ RETURNS ] return-type
  [ ( parameter [ , parameter ] ... ) ]
  IN hPortType .
```

function-name

The name of the function. You must avoid ABL reserved keywords. For a list of ABL keywords, see Keyword Index.

[RETURNS] *return-type*

Indicates the data type of the function return value. You can specify *return-type* as one of the following data types. For more information on each data type, see the Data types reference entry:

```
{ CHARACTER | COM-HANDLE | DATE | DATETIME | DATETIME-TZ
  | DECIMAL | HANDLE | INT64 | INTEGER | LOGICAL | LONGCHAR
  | MEMPTR | RAW | RECID | ROWID
  | [ CLASS ] object-type-name } [ EXTENT [ constant ] ]
```

object-type-name

Specifies the type name of an ABL or .NET class or interface. Specify an object type name using the syntax described in the Type-name syntax reference entry. With an appropriate USING statement, you can also specify a class or interface name alone, without the qualifying package or namespace.

CLASS

If the specified class or interface type name conflicts with an abbreviation of a built-in primitive type name, such as INT for INTEGER, you must specify the CLASS keyword.

For a class or interface return value, ABL returns an object reference associated with the class or interface, not a class instance itself. For more information on object references, see the Class-based object reference reference entry.

EXTENT [*constant*]

Defines the return value as an array of data elements with the specified primitive or object type. This option can specify an array return value as either determinate (has a defined number of elements) or indeterminate (has an undefined number of elements). For more information about determinate and indeterminate arrays, see Work with one-dimensional arrays in *Basic ABL*.

If you do not use the EXTENT option (or you specify *constant* as 0), the return value is not an array return value.

Note: If you invoke a function on an application server, the function cannot return a value as a LONGCHAR, MEMPTR, or CLASS.

PRIVATE

Indicates the following about the user-defined function:

- That it cannot be invoked from an external procedure—that is, from a procedure file external to the current procedure file.
- That the INTERNAL-ENTRIES attribute on the procedure that defines it does not provide its name (unless the procedure that defines it is the current procedure file).
- That the GET-SIGNATURE( ) method on the procedure that defines it does not provide its signature (unless the procedure that defines it is the current procedure file).

```
( parameter [ , parameter ] ... )
```

Defines one or more parameters of the function.

For information on the parameter definition syntax, see the Parameter definition syntax reference entry.

function-body

The body of a function definition. Define *function-body* using the following syntax:

```
function-logic
      .
      .
      .
    [ catch-block [ catch-block ... ] ]
    [ finally-block ]
END [ FUNCTION ] .
```

function-logic

The logic of the function. This logic can contain the ABL statements allowed within a procedure block, except that the RETURN ERROR statement returns the Unknown value (?) for the function, regardless of its return type, but does **not** raise ERROR in the caller.

To return a function value of the data type specified by *return-type*, you can execute the RETURN statement to set a value of that data type to return at run time. If you omit the RETURN statement, the function returns the Unknown value (?), regardless of the data type specified by *return-type*.

If *return-type* is defined as a .NET array of mapped types (for example, "System.Byte[]"), you must return an object reference of the specified .NET array of mapped types in the RETURN statement. You cannot return an ABL array of a type that maps to the .NET array type (for example, INTEGER EXTENT) or the AVM raises a run-time error. If you do not execute any RETURN statement for *return-type* in the *function-logic*, the user-defined function returns the Unknown value (?) as its return value.

Each logic statement must end with a period.

catch-block

Specifies a CATCH statement that defines error handling code for one or more error types. A DO block does not have any default error handling. Therefore, a DO block must have error handling options specified such that it becomes an undoable block. Otherwise, ABL generates a compiler warning. For more information on *catch-block*, see the CATCH statement reference entry.

finally-block

Specifies a FINALLY statement that defines the processing that must occur after all other processing in the block occurs. For more information on *finally-block*, see the FINALLY statement reference entry.

END [FUNCTION]

Specifies the end of the function body definition. You must end the function body definition with the END statement.

FORWARD

Declares a prototype for a function in a procedure whose definition appears later in the same procedure. You must declare a user-defined function prototype when the function definition appears within the same procedure following the first use of the function. This prototype must appear in the procedure before the first use of the function.

The FUNCTION statement with the FORWARD option must include the following information on the function: the data type it returns, and the data type and mode (INPUT, OUTPUT, or INPUT-OUTPUT) of each parameter.

If you declare a function prototype, reference it, and do not define it before the end of the procedure, the compiler returns an error.

[MAP [TO] *actual-name*] IN *proc-handle*

Declares prototype for a function that resides in a procedure external to the declaring procedure, with the following information:

- Optionally, that *function-name* (the second element in the FUNCTION statement) is an alias (alternative name) for the function and that *actual-name* is the name that appears in the function definition.
- The definition of the function resides in another external procedure specified by *proc-handle*, which represents an expression that evaluates to a handle to the procedure that defines the function. This procedure can be an active procedure in the local context or in a remote persistent, single-run, or singleton procedure.

Note: The MAP option might simplify your code if it references two different user-defined functions that have the same name but that reside in different procedures.

A FUNCTION statement with the IN *proc-handle* option must include the following information on the function: the data type it returns, and the data type and mode (INPUT, OUTPUT, or INPUT-OUTPUT) of each parameter.

IN SUPER

Declares a prototype for a function whose definition resides in a super procedure.

A FUNCTION statement with the IN SUPER option must include the following information on the function: the data type it returns, and the data type and mode (INPUT, OUTPUT, or INPUT-OUTPUT) of each parameter.

operationName

The name of a Web service operation specified in a WSDL file.

hPortType

A handle to a procedure object that encapsulates a Web service operation.

Examples

The first example, r-udf1.p, defines and references the user-defined function doubler( ), which accepts an integer and returns the integer multiplied by two:

r-udf1.p

```
/* r-udf1.p */
/* Defines and references a user-defined function */

/* Define doubler() */
FUNCTION doubler RETURNS INTEGER (INPUT parm1 AS INTEGER):
  RETURN (2 * parm1).
END FUNCTION.

/* Reference doubler() */
DISPLAY  "doubler(0)=" doubler(0) SKIP
  "doubler(1)=" doubler(1) skip
  "doubler(2)=" doubler(2) skip.
```

The second example, r-udf2.p, declares a prototype for, references, and defines doubler( ):

r-udf2.p

```
/* r-udf2.p */
/* Forward-declares, references, and defines a user-defined function */

/* Forward declare doubler() */
FUNCTION doubler RETURNS INTEGER (INPUT parm1 AS INTEGER) FORWARD.

/* Reference doubler() */
DISPLAY "doubler(0)=" doubler(0).
DISPLAY "doubler(1)=" doubler(1).
DISPLAY "doubler(2)=" doubler(2).

/* Define doubler() */
FUNCTION doubler RETURNS INTEGER (INPUT parm1 AS INTEGER):
  RETURN (2 * parm1).
END FUNCTION.
```

The third example consists of two procedures, r-udf3.p and r-udfdef.p. The example illustrates defining a prototype for user-defined function that is defined in an external procedure.

The procedure, r-udf3.p, declares the prototype for doubler( ), runs r-udfdef.p persistently, invokes doubler( ), and deletes the persistent procedure:

r-udf3.p

```
/* r-udf3.p */
/* References an externally-defined user-defined function */

/* Define items */
DEFINE VARIABLE myhand AS HANDLE    NO-UNDO.
DEFINE VARIABLE mystr  AS CHARACTER NO-UNDO FORMAT "x(20)".

/* Forward declare doubler() */
FUNCTION doubler RETURNS INTEGER (INPUT parm1 AS INTEGER) IN myhand.

/* Run the procedure that defines doubler() */
RUN src\prodoc\langref\r-udfdef.p PERSISTENT SET myhand.

/* Reference doubler() */
DISPLAY "doubler(0)=" doubler(0) SKIP
        "doubler(1)=" doubler(1) SKIP
        "doubler(17)=" doubler(17) SKIP.

/* Delete the procedure that defines doubler */
DELETE PROCEDURE myhand.
```

The second procedure, `r-udfdef.p`, defines `doubler( )`:

r-udfdef.p

```
/* r-udfdef.p */
/* Defines user-defined function doubler() */

FUNCTION doubler RETURNS INTEGER (INPUT parm1 AS INTEGER):
  RETURN (2 * parm1).
END FUNCTION.
```

To start the third example, run `r-udf3.p` in the Procedure Editor.

In the fourth example, `r-fctrl2.p`, the user-defined function `fact( )` implements the factorial function, common in probability and statistics, and commonly notated `"!"` (`6! = 6 x 5 x 4 x 3 x 2 x 1; 100! = 100 x 99 x 98 x ... x 3 x 2 x 1`):

r-fctrl2.p

```
/* r-fctrl2.p */
/* Demonstrates user-defined function fact() */

DEFINE VARIABLE inp AS INTEGER LABEL "Input Value".

FUNCTION fact RETURNS INTEGER (INPUT val AS INTEGER):
  IF val LT 0 THEN RETURN 0.
  IF val LE 1 THEN RETURN 1.
  RETURN val * fact(val - 1).
END.

REPEAT:
  UPDATE inp WITH TITLE "Factorials".
  DISPLAY fact(inp) LABEL "Factorial".
END.
```

Notes

- You can terminate a FUNCTION statement with either a period (.) or a colon (:), but typically use a colon (:) for a function definition and a period (.) for a function prototype or to declare a Web service operation.
- Before you reference a user-defined function within a procedure, you must define it, declare its prototype, declare it as external (by using FUNCTION statement's IN option), or define it.
- You cannot define shared objects, work tables, temp-tables, or ProDataSet objects within a user-defined function.
- ABL implements scalar and array parameters of user-defined functions as NO-UNDO variables.
- A reference to a user-defined function must match the declared prototype or definition with respect to the return type, and with respect to the number, type, and mode of the parameters.
- When an ABL predicate (such as a WHERE clause) contains a user-defined function, the AVM evaluates the function once—when it opens the query or enters the FOR EACH block.
- When the AVM encounters a user-defined function declared externally that references a user-defined function declared externally that references a user-defined function declared externally, etc., the AVM tolerates up to 64 levels of indirection. At the 65th level, the AVM raises an error and returns the Unknown value (?).
- If a user-defined function has one or more buffer parameters and its definition resides in another procedure, the referencing procedure and the defining procedure must reside on the same machine. If a user-defined function does not have buffer parameters, the invoking procedure and the defining procedure can reside on different machines.
- When you invoke a user-defined function (or a built-in function), you do not need to assign the function's return value to a variable. That is, you can invoke a user-defined function as a statement, ignoring the return value. You might use this technique with a function that performs some action on a persistent object, such as a shared variable, when you want the action to occur and do not need to check the return value. For example:

  ```
  doubler(my-shared-variable).
  ```

- When you invoke a user-defined function, you may pass a TABLE, TABLE-HANDLE, DATASET, or DATASET-HANDLE parameter by value, by reference, or by binding using the BY-VALUE, BY-REFERENCE, or BIND keyword, respectively. For example:

  ```
  myfunc(OUTPUT TABLE tt BY-REFERENCE).
  ```

  For more information about passing these parameters by value, by reference, or by binding, see the Parameter passing syntax reference entry.

- To return an error to the caller from a user-defined function, you can:
  - Migrate from using a user-defined function defined within a procedure to using a method defined within a class.
  - Use the ROUTINE-LEVEL ON ERROR UNDO, THROW statement in conjunction with the THROW option of the UNDO statement or the ON ERROR phrase in the user-defined function block.
  - Use the THROW option of the UNDO statement or the ON ERROR phrase from a CATCH block (CATCH statement) within the user-defined function block.
  - Invoke the STOP statement to raise the STOP condition in the caller.

See also

DYNAMIC-FUNCTION function, METHOD statement, Parameter definition syntax, PROCEDURE statement, RETURN statement

GATEWAYS function

The GATEWAYS function has been replaced by the DATASERVERS function, which is exactly equivalent.

This function is supported only for backward compatibility.

Note: Does not apply to SpeedScript programming.

Syntax

```
GATEWAYS
```

GE or >= operator

Returns a TRUE value if the first of two expressions is greater than or equal to the second expression.

Syntax

```
expression { GE | >= } expression
```

expression

A constant, field name, variable name, or any combination of these. The expressions on either side of the GE or >= must be of the same data type, although one might be integer and the other decimal.

Example

This procedure displays item information for those items whose `OnHand` value is greater than or equal to 120:

r-ge.p

```
FOR EACH Item NO-LOCK WHERE Item.OnHand >= 120:
  DISPLAY Item.ItemNum Item.ItemName Item.OnHand.
END.
```

Notes

- By default, the AVM uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (`-cpcoll`) startup parameter take precedence over a collation specified for any database the AVM accesses during the session, except when the AVM uses or modifies pre-existing indexes. If you do not specify a collation with the `-cpcoll` startup parameter, the AVM

uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, the AVM uses the collation rules with the default name "basic" (which might or might not exist in the convmap.cp file).

- If either of the expressions is the Unknown value (?), then the result is the Unknown value (?); if both expressions are the Unknown value (?), then the result is TRUE.
- You can compare character strings with GE. Most character comparisons are case insensitive in ABL. That is, upper-case and lower-case characters have the same sort value. However, it is possible to define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If either *expression* is a field or variable defined as case sensitive, the comparison is case sensitive and "Smith" does not equal "smith".
- Characters are converted to their sort code values for comparison. Using the default case-sensitive collation table, all uppercase letters sort before all lowercase letters (for example, a is greater than Z, but less than b.) Note also that in character code uppercase A is less than [, \ , ^ , _, and ' , but lowercase a is greater than these.
- You cannot compare data of different DATE, DATETIME, and DATETIME-TZ data types to each other using GE. You must first convert different date and datetime data types to the same data type before doing a comparison between them.
- You can use GE to compare a LONGCHAR variable to another LONGCHAR or CHARACTER variable. The variable values are converted to -cpinternal for comparison and must convert without error, or the AVM raises a run-time error.
- You cannot use GE to compare any of the LOB data types, including BLOB or CLOB.
- You can use GE to compare two enums, including both ABL and .NET enums, as long as they are the same enum type. GE compares the underlying numerical values of the enumerators.

GENERATE-PBE-KEY function

Generates a password-based encryption key, based on the PKCS#5/RFC 2898 standard, and returns the key as a RAW value.

Syntax

```
GENERATE-PBE-KEY( password [ , salt ] )
```

password

The password (a binary value) to use in generating the encryption key. This value may be of type CHARACTER, LONGCHAR, RAW, or MEMPTR. If the password contains a CHARACTER or LONGCHAR value, the AVM converts it to UTF-8 (which ensures a consistent value regardless of code page settings) before using it to generate the encryption key. To avoid this automatic conversion, specify a RAW or MEMPTR value. If you specify the Unknown value (?), the result is the Unknown value (?).

salt

An optional RAW expression that evaluates to the salt value (a random series of 8 bytes) to use in generating the encryption key. If you specify the Unknown value (?), the current value of the

ENCRYPTION-SALT attribute is used. If no salt value is specified in the ENCRYPTION-SALT attribute, no salt value is used.

You can also use the GENERATE-PBE-SALT function to generate a salt value, which can help to ensure that the password key value is unique.

If specified, this salt value is combined with the password value and hashed some number of times to generate a password-based encryption key (using the algorithm specified by the PBE-HASH-ALGORITHM attribute and the number of iterations specified by the PBE-KEY-ROUNDS attribute).

Notes

- You are responsible for generating, storing, and transporting these values.
- The size of the generated encryption key is determined by the cryptographic algorithm specified by the SYMMETRIC-ENCRYPTION-ALGORITHM attribute.
- Before invoking this function, be sure to set the PBE-HASH-ALGORITHM attribute to the name of the hash algorithm to use.
- If you call this function multiple times with the same password string, hash algorithm, number of iterations, and salt value, the same binary key is generated each time.

See also

GENERATE-PBE-SALT function, GENERATE-RANDOM-KEY function, MESSAGE-DIGEST function, SECURITY-POLICY system handle

GENERATE-PBE-SALT function

Generates a random salt value (a series of 8 bytes) to use in generating an encryption key, and returns the salt value as a RAW value. Using a salt value can help to ensure that a password key value is unique.

Syntax

```
GENERATE-PBE-SALT
```

Notes

- This salt value is combined with a password value and hashed some number of times to generate a password-based encryption key (using the algorithm specified by the PBE-HASH-ALGORITHM attribute and the number of iterations specified by the PBE-KEY-ROUNDS attribute).
- You are responsible for generating, storing, and transporting this value.

See also

GENERATE-PBE-KEY function, MESSAGE-DIGEST function, SECURITY-POLICY system handle

GENERATE-RANDOM-KEY function

Generates a pseudorandom (rather than a truly random) series of bytes to use as an encryption key, and returns the key as a RAW value.

Syntax

```
GENERATE-RANDOM-KEY
```

Notes

- You are responsible for generating, storing, and transporting this value.
- The size of the generated encryption key is determined by the cryptographic algorithm specified by the SYMMETRIC-ENCRYPTION-ALGORITHM attribute.
- The Alternate Random Number Generator (`-rand`) startup parameter setting has no effect on this function.

See also

GENERATE-PBE-KEY function, SECURITY-POLICY system handle

GENERATE-UUID function

Generates a universally unique identifier (UUID), as a 16-byte RAW value.

Syntax

```
GENERATE-UUID
```

Example

The following code fragment illustrates how to use the GENERATE-UUID function:

```
DEFINE VARIABLE MyUUID     AS RAW       NO-UNDO.
DEFINE VARIABLE Base64UUID AS CHARACTER NO-UNDO.

ASSIGN
  MyUUID     = GENERATE-UUID
  Base64UUID = BASE64-ENCODE(MyUUID).
```

You can use the GENERATE-UUID function with the BASE64-ENCODE function to generate a UUID and convert it to use in a Base64 character index. You can also remove the two trailing Base64 pad characters to reduce the size of the UUID. For example:

```
SUBSTRING(BASE64-ENCODE(GENERATE-UUID), 1, 22)
```

See also

BASE64-ENCODE function, GUID function

GET statement

Returns one record for a previously opened query.

Syntax

```
GET { FIRST | NEXT | PREV | LAST | CURRENT } query
    [ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ]
    [ NO-WAIT ]
```

`FIRST` *query*

Finds the first record associated with the query. The query must have been previously opened in an OPEN QUERY statement. The order of the records is determined by the options specified in the Record phrase the OPEN QUERY statement.

`NEXT` *query*

Returns the first or next record associated with the query. The query must have been previously opened in an OPEN QUERY statement. The order of the records is determined by the options specified in the OPEN QUERY statement of the Record phrase.

`PREV` *query*

Returns the preceding or last record associated with the query. The query must have been previously opened in an OPEN QUERY statement. The order of the records is determined by the options specified in the OPEN QUERY statement of the Record phrase.

`LAST` *query*

Returns the last record associated with the query. The query must have been previously opened in an OPEN QUERY statement. The order of the records is determined by the options specified in the OPEN QUERY of the Record phrase.

`CURRENT` *query*

Refetches the current record or records associated with the query. The query must have been previously opened in an OPEN QUERY statement. If the query is a join, the AVM returns the current record for all tables in the join.

`SHARE-LOCK`

Specifies that the record is share locked. Overrides the default locking of the OPEN QUERY statement. This applies to all buffers in a join.

EXCLUSIVE-LOCK

Specifies that the record is exclusively locked. Overrides the default locking of the OPEN QUERY statement. This applies to all buffers in a join.

NO-LOCK

Specifies that no lock is applied to the record. Overrides the default locking of the OPEN QUERY statement. This applies to all buffers in a join.

NO-WAIT

Specifies that the GET statement returns immediately if the record cannot be accessed because it is locked by another user. If you do not use the NO-WAIT option, the GET statement waits until the record can be accessed. This applies to all buffers in a join. If you specify NO-WAIT and the record is locked by another user, the record is returned to you with NO-LOCK and the LOCKED function returns TRUE for the record.

Example

This procedure uses the GET statement to find Customer orders:

r-getord.p

```
DEFINE QUERY cust-order FOR Customer, Order.

OPEN QUERY cust-order FOR EACH Customer, EACH Order OF Customer.

GET FIRST cust-order.

DO WHILE AVAILABLE Customer:
  DISPLAY Customer.CustNum Customer.Name
    WITH FRAME cust-info.
  DISPLAY Order WITH FRAME order-info SIDE-LABELS.
  PAUSE.

  GET NEXT cust-order.
END.
```

In the example, the GET FIRST statement fetches the first Customer record and the first Order record for that Customer. The GET NEXT statement fetches the next Order record for the Customer. If no more Order records are found for the current Customer, then the GET NEXT statement fetches the next Customer and the first Order record for that Customer. If a Customer has no Orders, the GET statement skips that Customer.

Notes

- The query must be opened with the OPEN QUERY statement before any records are fetched.
- A query that references more than one buffer defines a join. Each GET statement returns one set of records.
- If you execute a GET NEXT statement after the last record of the query has been fetched or you execute a GET PREV statement after the first record of the query has been fetched, the ERROR condition is **not** raised. However, you **can** use the AVAILABLE function to test whether a record was returned for the query fetch. You can also use the QUERY-OFF-END function to determine if the query is positioned at the end of its result list.
- If the query is positioned before the first record, GET NEXT acts the same as a GET FIRST; similarly, if the query is positioned beyond the last record, GET PREV acts the same as GET LAST.

- The GET LAST statement can be slow unless the AVM has performed a presort or already returned the last record that satisfies the query, or you specify USE-INDEX for the query (or the query happens to only use one index). Also, GET LAST might be slow if the query involves an outer join.
- If you do not specify a lock type, the AVM uses the lock type specified in the OPEN QUERY statement. If no lock type is specified in either the GET or OPEN QUERY statement, then the default ABL locking rules apply.
- If a GET CURRENT statement fails because of a lock conflict, the AVM rereads the record with a NO-LOCK status.
- When a GET statement executes, any FIND triggers defined for the tables are executed.
- FIND triggers do not execute for a GET CURRENT statement.
- To upgrade the lock on only one table in a join, use the FIND CURRENT statement.
- A query that includes a BREAK BY phrase becomes a FORWARD-ONLY query. In this case you cannot use the GET FIRST, GET LAST, or GET PREV statements. If you do, the AVM raises ERROR.

See also

AVAILABLE function, CLOSE QUERY statement, CURRENT-CHANGED function, CURRENT-RESULT-ROW function, DEFINE QUERY statement, FIND statement, FOR statement, LOCKED function, NUM-RESULTS function, OPEN QUERY statement, QUERY-OFF-END function, REPOSITION statement

GET-BITS function

Interprets one or more consecutive bits in an integer variable or field as an ABL integer value and returns that value.

Syntax

```
GET-BITS( source , position , numbits )
```

source

An ABL integer variable.

position

A variable or expression that returns an integer. This parameter designates the position of the lowest-order bit of the bits that are to be interpreted as an integer. Bits are numbered from 1 through the length of an integer; with 1 being the low-order bit. If *position* is greater than the length of an integer, the AVM returns the Unknown value (?). If *position* is less than 1, the AVM generates a run-time error.

numbits

The number of bits to examine when generating the return value. If *position* plus *numbits* is greater than the length of an integer plus 1, the AVM generates a run-time error.

Note

This function can return a value greater than 32 bits if *source* is an INT64.

See also

PUT-BITS statement

GET-BYTE function

Returns the unsigned 1 byte value at the specified memory location as an INTEGER value.

Syntax

```
GET-BYTE ( source , position )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-BYTE returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, the AVM returns the Unknown value (?). If *position* is less than 1, the AVM generates a run-time error.

Examples

In this example, the RAW function goes to the Customer field in the non-OpenEdge database. The GET-BYTE function accesses the first byte and stores the integer value of that byte in the variable ix. The procedure then tests the value, if the integer value is 83 (the character code value for S), the AVM displays the Customer Name.

r-rawget.p

```
/* You must connect to a non-OpenEdge database to run this procedure */
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

FOR EACH Customer:
  ix = GET-BYTE(RAW(Customer.Name), 1).
  IF ix = 83 THEN
    DISPLAY Customer.Name.
END.
```

The next procedure sets up a MEMPTR region with a character string and uses the GET-BYTE function to display the character code value of each character in the string:

r-mptget.p

```
DEFINE VARIABLE mptr AS MEMPTR  NO-UNDO.
DEFINE VARIABLE cnt  AS INTEGER NO-UNDO.

ASSIGN
  SET-SIZE(mptr)      = LENGTH("DANIEL") + 1
  PUT-STRING(mptr, 1) = "DANIEL".

REPEAT cnt = 1 TO LENGTH("DANIEL"):
  DISPLAY GET-BYTE(mptr, cnt).
END.
```

Notes

- For more information on using the MEMPTR and RAW data types, see *OpenEdge Programming Interfaces*.
- You can use the alternative keyword GETBYTE instead of GET-BYTE.

See also

LENGTH function, PUT-BYTE statement, RAW function, RAW statement, SET-SIZE statement

GET-BYTE-ORDER function

Returns an INTEGER value indicating the byte order setting of a MEMPTR variable. This will be either the value provided by the last execution of SET-BYTE-ORDER with this MEMPTR variable, or HOST-BYTE-ORDER if SET-BYTE-ORDER has not been executed.

Syntax

```
GET-BYTE-ORDER( memptr )
```

memptr

An expression that returns a MEMPTR.

Note

GET-BYTE-ORDER never affects data currently in the MEMPTR. That is, it does not actually re-order the data.

See also

SET-BYTE-ORDER statement

GET-BYTES function

Returns the specified number of bytes, from the specified location, into a RAW or MEMPTR variable.

Syntax

```
GET-BYTES( source , position , numbytes )
```

source

An expression that evaluates to a RAW or MEMPTR value indicating the source location. If *source* is the Unknown value (?), GET-BYTES returns the Unknown value (?).

position

An integer value greater than 0 indicating the byte position of the first byte to get. If *position* is greater than the length of *source*, the AVM returns the Unknown value (?). If *position* is less than 1, the AVM generates a run-time error.

numbytes

An integer value greater than 0 indicating how many bytes to return as a RAW value. If *position* plus *numbytes* is greater than the size of *source*, the AVM returns the Unknown value (?).

If the variable that accepts the returned data is a RAW variable and *numbytes* is greater than its length but less than or equal to 32K, the AVM increases the size of the variable to *numbytes*.

If either the source location, *source*, or the variable that accepts the returned data is a RAW value, and *numbytes* is greater than 32K, the AVM generates a run-time error.

If the variable that accepts the returned data is a MEMPTR variable and *numbytes* is greater than its length, the AVM generates a run-time error.

See also

PUT-BYTES statement

GET-CLASS function

Returns the object reference for the Progress.Lang.Class instance associated with the specified class or interface type.

Syntax

```
GET-CLASS ( object-type-name )
```

object-type-name

Specifies the type name of an ABL or .NET class or interface type, using the syntax described in the Type-name syntax reference entry. With an appropriate USING statement, you can also specify an unqualified class or interface name alone. If *object-type-name* is not the name of an existing ABL class that can be found on PROPATH or (for Windows) a .NET class that can be resolved based on the assemblies.xml file in use, a compiler error will result.

Example

Unlike the GetClass() method of Progress.Lang.Class, GET-CLASS takes a type name as its argument instead of a CHARACTER expression that holds a type name. As a result, USING statements can be applied to the argument, and the compiler checks the type name at compile time.

The following code uses GET-CLASS to determine if acme.myObjs.CustObj is FINAL:

```
USING Progress.Lang.*.
USING acme.myObjs.*.

DEFINE VARIABLE lFinal AS LOGICAL.

lFinal = GET-CLASS(CustObj):IsFinal().
MESSAGE "Is the class FINAL? " lFinal VIEW-AS ALERT-BOX.
```

See also

GetClass( ) method (Progress.Lang.Object)

GET-CODEPAGE function

The GET-CODEPAGE function returns the code page of a LONGCHAR variable or CLOB field.

Syntax

```
GET-CODEPAGE ( large-char-object )
```

large-char-object

The name of a LONGCHAR variable or CLOB field. If the specified LONGCHAR is empty and the code page was not fixed using the FIX-CODEPAGE statement, the AVM returns the Unknown value (?).

See also

FIX-CODEPAGE statement, IS-CODEPAGE-FIXED function

GET-CODEPAGES function

The GET-CODEPAGES function returns a comma-delimited list of the code pages listed in `convmap.cp` or specified by the Conversion Map (`-convmap`) startup parameter for the current ABL session.

Syntax

```
GET-CODEPAGES
```

Example

This procedure displays a list of the code pages available in memory for the current ABL session and the collations available for each code page:

r-get.p

```
DEFINE VARIABLE code-page-list AS CHARACTER NO-UNDO.
DEFINE VARIABLE collation-list AS CHARACTER NO-UNDO.
DEFINE VARIABLE ix             AS INTEGER   NO-UNDO.
DEFINE VARIABLE jx             AS INTEGER   NO-UNDO.

code-page-list = GET-CODEPAGES.

REPEAT ix = 1 TO NUM-ENTRIES(code-page-list):
  DISPLAY ENTRY(ix, code-page-list) FORMAT "x(19)" COLUMN-LABEL "Code Page"
    WITH DOWN FRAME a.
  collation-list = GET-COLLATIONS(ENTRY(ix, code-page-list)).

  REPEAT jx = 1 TO NUM-ENTRIES(collation-list):
    DISPLAY ENTRY(jx, collation-list) FORMAT "x(19)"
      COLUMN-LABEL "Collation"
      WITH DOWN FRAME a.
    DOWN WITH FRAME a.
  END.
END.
```

See also

GET-COLLATIONS function

GET-COLLATION function

The GET-COLLATION function returns the collation name for a CLOB field.

Syntax

```
GET-COLLATION ( clob-field )
```

clob-field

A CLOB field name.

GET-COLLATIONS function

The GET-COLLATIONS function returns a comma-delimited list of the collations either listed in `convmap.cp` or specified by the Conversion Map (`-convmap`) startup parameter for the specified code page.

Syntax

```
GET-COLLATIONS ( codepage )
```

codepage

A code page name. If there are no collations for the specified code page, the AVM returns the Unknown value (?).

Example

This procedure displays a list of the code pages available in memory for the current ABL session and the collations available for each code page:

r-get.p

```
DEFINE VARIABLE code-page-list AS CHARACTER NO-UNDO.
DEFINE VARIABLE collation-list AS CHARACTER NO-UNDO.
DEFINE VARIABLE ix             AS INTEGER   NO-UNDO.
DEFINE VARIABLE jx             AS INTEGER   NO-UNDO.

code-page-list = GET-CODEPAGES.

REPEAT ix = 1 TO NUM-ENTRIES(code-page-list):
  DISPLAY ENTRY(ix, code-page-list) FORMAT "x(19)" COLUMN-LABEL "Code Page"
    WITH DOWN FRAME a.
  collation-list = GET-COLLATIONS(ENTRY(ix, code-page-list)).

  REPEAT jx = 1 TO NUM-ENTRIES(collation-list):
    DISPLAY ENTRY(jx, collation-list) FORMAT "x(19)"
      COLUMN-LABEL "Collation"
      WITH DOWN FRAME a.
    DOWN WITH FRAME a.
  END.
END.
```

See also

GET-CODEPAGES function

GET-DB-CLIENT function

Returns the handle to a copy of the sealed client-principal object that represents the user identity for the specified database connection.

Syntax

```
GET-DB-CLIENT ( [db-exp] )
```

db-exp

An optional character expression that evaluates to a case-insensitive logical or alias name of an OpenEdge RDBMS. This expression can be unspecified or evaluate to the Unknown value (?) only if there is a single OpenEdge database connection, in which case the client-principal object handle is returned for that connection.

Example

In the following procedure fragment, GET-DB-CLIENT returns the sealed client-principal that the CONNECT statement creates from authenticating a new connection to the `Sports2020` database based on the user ID (`cUserID`) and password (`cPasswd`) that are passed as input parameters:

```
DEFINE INPUT PARAMETER cUserID AS CHARACTER NO-UNDO.
DEFINE INPUT PARAMETER cPasswd AS CHARACTER NO-UNDO.

DEFINE VARIABLE hCP AS HANDLE NO-UNDO.

CONNECT C:\OpenEdge\WRK\db\Sports2020
  VALUE( "-U " + cUserID +
         " -P " + "oech1::" + AUDIT-POLICY:ENCRYPT-AUDIT-MAC-KEY(cPasswd))
  -H dbserver -S 1900 NO-ERROR.
ASSIGN hCP = GET-DB-CLIENT("Sports2020").

SECURITY-POLICY:LOAD-DOMAINS("Sports2020").
SECURITY-POLICY:SET-CLIENT(hCP).

DELETE OBJECT hCP.
ASSIGN hCP = ?.
```

The fragment then loads the ABL session domain registry from the `Sports2020` database and uses the returned client-principal object (hCP) to set the session identity (and the identities of any other available database connections) to the existing `Sports2020` connection identity. The fragment ends by deleting the client-principal, which is no longer needed for the session.

Typically, you also check the LOGIN-STATE attribute and STATE-DETAIL attribute on the client-principal object handle, along with other error handling mechanisms, to identity if both the database connection and its authentication are successful.

Note also that this fragment encrypts the password value (`cPasswd`) and concatenates it with a prefix in a form that OpenEdge expects for encrypted passwords. For more information, see the ENCRYPT-AUDIT-MAC-KEY( ) method reference entry.

Notes

- You can use the client-principal object returned by this function to set the identity for this and other database connections or ABL sessions using single sign-on (SSO) operations **unless** the object represents an OpenEdge default connection identity. A default connection identity is set by establishing a database connection **without** specifying the User ID (`-U`) and Password (`-P`) connection parameters, either on the AVM startup command line or as options of the CONNECT statement.

To seal a client principal with a default connection identity, OpenEdge creates a unique domain access code to seal the client-principal object returned by this function. Sealing the client principal with an internal domain access code provides backwards compatibility with previous OpenEdge releases that prohibit a database connection from being reverted back to the connection's default user identity. Thus, you can use the GET-DB-CLIENT function to return a valid client-principal object from a database connection with a default connection identity. However, because the domain access code used to seal the object is not configured for any registered domain, you cannot use that client-principal in single sign-on (SSO) operations to assign a default user identity to any ABL session or database connection, including the connection for which GET-DB-CLIENT returned the client-principal.

For more information on how OpenEdge sets a default connection identity, see the reference entry for the User ID (`-U`) client connection parameter in *Startup Command and Parameter Reference*.

- To avoid a memory leak, you must explicitly delete the client-principal object whose handle is returned by this function when you no longer need it.

See also

Client-principal object handle, CONNECT statement, GET-CLIENT( ) method, SET-DB-CLIENT function, SETUSERID function

---

GET-DOUBLE function

Returns the 8-byte floating-point value at the specified memory location as a DECIMAL value.

Syntax

```
GET-DOUBLE ( source , position )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-DOUBLE returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, the AVM returns the Unknown value (?). If *position* is less than 1, the AVM generates a run-time error.

Example

For examples of how to use the GET-DOUBLE function, see the GET-BYTE function reference entry.

Notes

- This function supports byte-swapping only if *source* is a MEMPTR data type. The function will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately before interpreting them. The AVM does not swap the bytes in the MEMPTR's memory, but does the byte-swap as it creates the return value.
- For more information on using the MEMPTR and RAW data types, see *OpenEdge Programming Interfaces*.

See also

LENGTH function, PUT-DOUBLE statement, RAW function, RAW statement, SET-SIZE statement

---

GET-EFFECTIVE-TENANT-ID function

Returns the tenant ID of the effective tenant as an integer.

Syntax

```
GET-EFFECTIVE-TENANT-ID( [ database-name ] )
```

database-name

A character expression that evaluates to a logical database name or database alias. If no database is specified and more than one database is connected, the AVM raises an error. If the database name is not a valid name for a connected database, the AVM raises an error.

Notes

- If a super tenant executes this function and they have already executed the SET-EFFECTIVE-TENANT function, this function returns the tenant ID of any effective tenant that is in scope. If they have not executed the SET-EFFECTIVE-TENANT function or there is no effective tenant in scope, this function returns the default tenant ID (0).
- If a regular-tenant user executes this function, it returns the tenant ID associated with the user's tenancy.

See also

GET-EFFECTIVE-TENANT-NAME function, SET-EFFECTIVE-TENANT function, TENANT-ID function

---

GET-EFFECTIVE-TENANT-NAME function

Returns the name of the effective tenant as a character string.

Syntax

```
GET-EFFECTIVE-TENANT-NAME( [ database-name ] )
```

database-name

A character expression that evaluates to a logical database name or database alias. If no database is specified and more than one database is connected, the AVM raises an error. If the database name is not a valid name for a connected database, the AVM raises an error.

Notes

- If a super tenant executes this function and they have already executed the SET-EFFECTIVE-TENANT function, this function returns the name of any effective tenant that is in scope. If they have not executed the SET-EFFECTIVE-TENANT function or there is no effective tenant in scope, this function returns the name of the default tenant (`"Default"`).
- If a regular-tenant user executes this function, it returns the tenant name associated with the user's tenancy.

See also

GET-EFFECTIVE-TENANT-ID function, SET-EFFECTIVE-TENANT function, TENANT-NAME function

---

GET-FLOAT function

Returns the 4-byte floating-point value at the specified memory location as a DECIMAL value.

Syntax

```
GET-FLOAT ( source , position )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-FLOAT returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, the AVM returns the Unknown value (?). If *position* is less than 1, the AVM generates a run-time error.

Examples

For examples of how to use the GET-FLOAT function, see the GET-BYTE function reference entry.

Notes

- This function supports byte-swapping only if *source* is a MEMPTR data type. The function will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately before interpreting them. The AVM does not swap the bytes in the MEMPTR's memory, but does the byte-swap as it creates the return value.
- For more information on using the MEMPTR and RAW data types, see *OpenEdge Programming Interfaces*.

See also

LENGTH function, PUT-FLOAT statement, RAW function, RAW statement, SET-SIZE statement

GET-INT64 function

Returns the signed 64-bit value at the specified memory location as an INT64 value.

Syntax

```
GET-INT64 ( source , position )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-INT64 returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, the AVM returns the Unknown value (?). If *position* is less than 1, the AVM generates a run-time error.

Example

The following example demonstrates how to get values in and out of RAW and MEMPTR values:

r-getint64.p

```
DEFINE VARIABLE myint64  AS INT64   NO-UNDO INITIAL 7888999000.
DEFINE VARIABLE myint    AS INTEGER NO-UNDO INITIAL 2345.
DEFINE VARIABLE myraw    AS RAW     NO-UNDO.
DEFINE VARIABLE result64 AS INT64   NO-UNDO.
DEFINE VARIABLE result1  AS INTEGER NO-UNDO.
DEFINE VARIABLE mymem    AS MEMPTR  NO-UNDO.

ASSIGN
  PUT-LONG(myraw,1)  = myint
  result1            = GET-LONG(myraw,1)
  PUT-INT64(myraw,1) = myint64
  result64           = GET-INT64(myraw,1).

MESSAGE "raw version " result1 result64.

ASSIGN
  result1            = ?
  result64           = ?
  SET-SIZE(mymem)    = 40
  PUT-LONG(mymem,1)  = myint
  result1            = GET-LONG(mymem,1)
  PUT-INT64(mymem,1) = myint64
  result64           = GET-INT64(mymem,1).

MESSAGE "memptr version " result1 result64.

result1 = get-int64(mymem,1) NO-ERROR.

MESSAGE "store getint64 in int gives" ERROR-STATUS:GET-MESSAGE(1).
MESSAGE "doing put-long of in64" myint64.

PUT-LONG(mymem,1) = myint64 NO-ERROR.
```

```
MESSAGE "storeint64 with put-long gives" ERROR-STATUS:GET-MESSAGE(1).
SET-SIZE(mymem) = 0.
```

See also

INT64 function, INTEGER function, PUT-INT64 statement, GET-LONG function

GET-KEY-VALUE statement
(Windows only)

Searches the current environment for a particular key and places its value into a particular data item.

Note: Does not apply to SpeedScript programming.

Syntax

```
GET-KEY-VALUE SECTION section-name
  KEY { key-name | DEFAULT }
  VALUE key-value
```

`SECTION` *`section-name`*

A CHARACTER expression that specifies the name of the section that contains the key of interest.

In initialization files, section names appear in square brackets([]). When you specify a section name in the GET-KEY-VALUE statement, omit the square brackets.

`KEY` *`key-name`*

A CHARACTER expression that specifies the name of the key of interest.

If you specify the Unknown value (?) or the empty string (""), GET-KEY-VALUE returns a comma-separated list of all keys in the section you specified.

`DEFAULT`

Tells GET-KEY-VALUE to use the default key of section *`section-name.`*

Some applications store data in the registry under the default key of a section. This option lets you retrieve this data. For an example, see the EXAMPLES section of this entry.

This option applies only to the registry and not to initialization files.

`VALUE` *`key-value`*

The name of a CHARACTER variable to hold the value of the key of interest.

Examples

If the current environment resides in the registry, the GET-KEY-VALUE statement:

1. Searches the current environment for the subkey MYSECTION
2. Searches MYSECTION for the value name MYKEY
3. Assigns the value of MYKEY to the variable MYVARIABLE

If the current environment resides in an initialization file, the GET-KEY-VALUE statement:

1. Searches the section MYSECTION for the key MYKEY
2. Assigns the value of MYKEY to the variable MYVARIABLE as shown in the following example:

```
GET-KEY-VALUE SECTION "MYSECTION" KEY "MYKEY" VALUE MYVARIABLE
```

If the current environment is the registry, the GET-KEY-VALUE statement:

1. Searches the current environment for the key MYKEY
2. Assigns the value of MYKEY to the variable MYVARIABLE

If the current environment resides in an initialization file, the following example returns a comma-separated list of all section names in the initialization file:

```
GET-KEY-VALUE SECTION "" KEY "MYKEY" VALUE MYVARIABLE
```

If the current environment resides in the registry, the GET-KEY-VALUE statement:

1. Searches the current environment for the subkey MYSECTION
2. Returns a comma-separated list of all value names in MYSECTION

If the current environment resides in an initialization file, the GET-KEY-VALUE statement:

1. Searches the current environment for the section MYSECTION
2. Returns a comma-separated list of all key names in MYSECTION; for example:

```
GET-KEY-VALUE SECTION "MYSECTION" KEY "" VALUE MYVARIABLE
```

```
GET-KEY-VALUE SECTION "MYSECTION" KEY "?" VALUE MYVARIABLE
```

If the current environment resides in the registry, the following examples return a comma-separated list of subkeys under the current environment location and all value names directly under the current environment location. The delimiter @value@ separates the subkey names from the value names.

If the current environment resides in an initialization file, the following examples return a comma-separated list of all section names in the initialization file:

```
GET-KEY-VALUE SECTION "" KEY "" VALUE MYVARIABLE
```

```
GET-KEY-VALUE SECTION "" KEY "?" VALUE MYVARIABLE
```

```
GET-KEY-VALUE SECTION "?" KEY "" VALUE MYVARIABLE
```

```
GET-KEY-VALUE SECTION "?" KEY "?" VALUE MYVARIABLE
```

If the current environment resides in the registry, the GET-KEY-VALUE statement:

1. Searches the current environment for the subkey MYAPP
2. Assigns the value of the default key under MYAPP to the variable MYVARIABLE

If the current environment resides in an initialization file, the following example returns an error:

```
GET-KEY-VALUE SECTION "MYAPP" KEY DEFAULT VALUE MYVARIABLE
```

Notes

- Environments typically consist of sections, each of which can contain keys, each of which consists of a name and a value. A typical section name is COLORS. A typical key within this section consists of the name "COLOR16" and the value 255,255,0. This key attaches this particular name to this particular color. (The value represents a color specification using the red-green-blue color-naming scheme.)

  The current environment might be the registry or an initialization file. The registry consists of sections called keys and subkeys arranged in a hierarchy. Keys and subkeys contain value entries, each of which consists of a value name and value data. Initialization files, by contrast, consist of a single level of sections. Sections contain entries, each of which consists of a name, an equals sign (=), and a value.

  For more information on environments, see the topics on colors and fonts in *OpenEdge Programming Interfaces*.

- The current environment is either the default environment, the **startup environment** (an environment that a startup parameter specified), or an application environment that the LOAD statement loaded and that the USE statement made current.
- If you unload the current environment (using the UNLOAD statement) and then use the GET-KEY-VALUE statement, you access the startup environment.

See also

LOAD statement, PUT-KEY-VALUE statement, UNLOAD statement, USE statement

GET-LONG function

Returns the signed 32-bit value at the specified memory location as an INTEGER value.

Syntax

```
GET-LONG ( source , position )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-LONG returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, the AVM returns the Unknown value (?). If *position* is less than 1, the AVM generates a run-time error.

Examples

For examples of how to use the GET-LONG function, see the GET-BYTE function reference entry.

Notes

- This function supports byte-swapping only if *source* is a MEMPTR data type. The function will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately before interpreting them. The AVM does not swap the bytes in the MEMPTR's memory, but does the byte-swap as it creates the return value.
- For more information on using the MEMPTR and RAW data types, see *OpenEdge Programming Interfaces*.

See also

LENGTH function, PUT-LONG statement, RAW function, RAW statement, SET-SIZE statement

GET-POINTER-VALUE function

Returns, as an INT64 value, the address of (or pointer to) the memory region associated with the specified MEMPTR variable. The returned value is based on whether the platform supports 64-bit pointers or 32-bit pointers. On a 32-bit platform, the value never gets bigger than 2GB.

Note: Does not apply to SpeedScript programming.

Syntax

```
GET-POINTER-VALUE ( memptr-var )
```

memptr-var

A reference to a variable defined as MEMPTR. If the variable is uninitialized (has no associated memory region), the function returns 0.

Example

This function is particularly useful when building a structure in an MEMPTR region that references other MEMPTR regions. It allows you to obtain the pointer to one MEMPTR region and store it in the structure you create in another MEMPTR region. The following example allocates three memory regions-for a BITMAPINFO structure, a BITMAPINFOHEADER structure, and an RGB color array. It then uses the GET-POINTER-VALUE function together with the PUT-LONG statement to store pointers to the BITMAPINFOHEADER structure and an RGB color array in the BITMAPINFO structure. These structures describe a device-independent bitmap for Windows dynamic link library (DLL) routines. For more information on these bitmap structures, see your Windows Software Development Kit documentation.

Note: The following example only works on a 32-bit machine because it leaves space for 4 bytes for the pointer values. On a 64-bit machine, you would have to allocate 8 bytes of space.

r-ptrval.p

```
DEFINE VARIABLE bitmapinfo       AS MEMPTR NO-UNDO.
DEFINE VARIABLE bitmapinfoheader AS MEMPTR NO-UNDO.
DEFINE VARIABLE RGBcolors        AS MEMPTR NO-UNDO.

SET-SIZE(bitmapinfo)       = 4  /* Pointer to bitmapinfoheader */
                           + 4. /* Pointer to RGBcolors        */

SET-SIZE(bitmapinfoheader) = 4  /* biSize          */
                           + 4  /* biWidth         */
                           + 4  /* biHeight        */
                           + 2  /* biPlanes        */
                           + 2  /* biBitCount      */
                           + 4  /* biCompression   */
                           + 4  /* biSizeImage     */
                           + 4  /* biXpelsPerMeter */
                           + 4  /* biYPelsPerMeter */
                           + 4  /* biClrUsed       */
                           + 4. /* biClrImportant  */

SET-SIZE(RGBcolors)        = 16 * 4. /* Array for 16 RGB color values */

/* Initialize pointers to bit map info header and RGB color array */
PUT-LONG(bitmapinfo,1) = GET-POINTER-VALUE(bitmapinfoheader).
PUT-LONG(bitmapinfo,5) = GET-POINTER-VALUE(RGBcolors).
```

Note: Before using structures such as these, you must initialize them according to your DLL requirements. For example, the biBitCount segment of the bitmapinfoheader must be set to 4 to specify the number of possible colors available in the RGB color array (16).

Notes

- MEMPTR structures are initialized using the SET-SIZE statement.
- For more information on using the MEMPTR data type, see *OpenEdge Programming Interfaces*.

See also

PUT-INT64 statement, PUT-LONG statement, SET-SIZE statement

GET-SHORT function

Returns the signed 16-bit value at the specified memory location as an INTEGER value.

Syntax

```
GET-SHORT ( source , position )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-SHORT returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, the AVM returns the Unknown value (?). If *position* is less than 1, the AVM generates a run-time error.

Examples

For examples of how to use the GET-SHORT function, see the GET-BYTE function reference entry.

Notes

- This function supports byte-swapping only if *source* is a MEMPTR data type. The function will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately before interpreting them. The AVM does not swap the bytes in the MEMPTR's memory, but does the byte-swap as it creates the return value.
- For more information on using the MEMPTR and RAW data types, see *OpenEdge Programming Interfaces*.

See also

LENGTH function, PUT-SHORT statement, RAW function, RAW statement, SET-SIZE statement

GET-SIZE function

Returns, as an INT64 value, the allocated byte size of the memory region associated with the specified MEMPTR variable.

Note: Does not apply to SpeedScript programming.

Syntax

```
GET-SIZE ( memptr-var )
```

memptr-var

A MEMPTR variable. If the variable is uninitialized (has no associated memory region), the function returns 0.

Example

The following example allocates three memory regions-for a BITMAPINFO structure, a BITMAPINFOHEADER structure, and an RGB color array. It then displays the allocation size for each region. These structures describe a device-independent bitmap for Windows dynamic link library (DLL) routines. For more information on these bitmap structures, see your Windows Software Development Kit documentation.

r-getsiz.p

```
DEFINE VARIABLE bitmapinfo       AS MEMPTR NO-UNDO.
DEFINE VARIABLE bitmapinfoheader AS MEMPTR NO-UNDO.
DEFINE VARIABLE RGBcolors        AS MEMPTR NO-UNDO.

SET-SIZE(bitmapinfo)       = 4  /* Pointer to bitmapinfoheader */
                           + 4. /* Pointer to RGBcolors        */

SET-SIZE(bitmapinfoheader) = 4  /* biSize          */
                           + 4  /* biWidth         */
                           + 4  /* biHeight        */
                           + 2  /* biPlanes        */
                           + 2  /* biBitCount      */
                           + 4  /* biCompression   */
                           + 4  /* biSizeImage     */
                           + 4  /* biXpelsPerMeter */
                           + 4  /* biYPelsPerMeter */
                           + 4  /* biClrUsed       */
                           + 4  /* biClrImportant  */
                            .

SET-SIZE(RGBcolors)        = 16 * 4. /* Array for 16 RGB color values */

DISPLAY
  GET-SIZE(bitmapinfo)
    LABEL "Bitmap info structure size" COLON 30 SKIP
  GET-SIZE(bitmapinfoheader)
    LABEL "Bitmap header structure size" COLON 30 SKIP
  GET-SIZE(RGBcolors)
    LABEL "Bitmap colors array size" COLON 30
  WITH SIDE-LABELS.
```

Notes

- To return a memory size greater than 0, the MEMPTR variable must be fully initialized, not just pre-initialized by a DLL or UNIX shared library routine.
- MEMPTR structures are initialized using the SET-SIZE statement.
- For more information on using the MEMPTR data type, see *OpenEdge Programming Interfaces*.

See also

SET-SIZE statement

GET-STRING function

Returns the null-terminated character string at the specified memory location as a CHARACTER value (not including the null terminator) or the number of bytes specified starting from the specified memory location as a CHARACTER value.

Syntax

```
GET-STRING ( source , position [ , numbytes ] )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-STRING returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, the AVM returns the Unknown value (?). If *position* is less than 1, the AVM generates a run-time error.

numbytes

An integer value greater than 0 that indicates how many bytes to convert into the CHARACTER value that is returned. If *numbytes* is not specified, or is -1, GET-STRING( ) returns all bytes until it encounters a NULL value.

Examples

For examples of how to use the GET-STRING function, see the GET-BYTE function reference entry.

Note

For more information on using the MEMPTR and RAW data types, see *OpenEdge Programming Interfaces*.

See also

LENGTH function, PUT-STRING statement, RAW function, RAW statement, SET-SIZE statement

GET-UNSIGNED-LONG function

Returns the unsigned 32-bit value at the specified memory location as an INT64. This is analogous to the GET-UNSIGNED-SHORT function, except with a 32-bit value.

Syntax

```
GET-UNSIGNED-LONG ( source , position )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-UNSIGNED-LONG returns the Unknown value (?).

position

An INTEGER or INT64 value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, the AVM returns the Unknown value (?). If *position* is less than 1, the AVM generates a run-time error.

Notes

- When returning the value from GET-UNSIGNED-LONG() to an INTEGER, if the value exceeds the maximum value of an INTEGER, the AVM generates a run-time error.
- This function supports byte-swapping only if *source* is a MEMPTR data type. The function will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately before interpreting them. The AVM does not swap the bytes in the MEMPTR's memory, but does the byte-swap as it creates the return value.
- For more information on using the MEMPTR and RAW data types, see *OpenEdge Programming Interfaces*.

See also

LENGTH function, PUT-UNSIGNED-LONG statement, RAW function, RAW statement, SET-SIZE statement

GET-UNSIGNED-SHORT function

Returns the unsigned 16-bit value at the specified memory location as an INTEGER value.

Syntax

```
GET-UNSIGNED-SHORT ( source , position )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-UNSIGNED-SHORT returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, the AVM returns the Unknown value (?). If *position* is less than 1, the AVM generates a run-time error.

Examples

For examples of how to use the GET-UNSIGNED-SHORT function, see the GET-BYTE function reference entry.

Notes

- This function supports byte-swapping only if *source* is a MEMPTR data type. The function will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately before interpreting them. The AVM does not swap the bytes in the MEMPTR's memory, but does the byte-swap as it creates the return value.
- For more information on using the MEMPTR and RAW data types, see *OpenEdge Programming Interfaces*.

See also

LENGTH function, PUT-SHORT statement, RAW function, RAW statement, SET-SIZE statement

GO-PENDING function

Returns a TRUE value if, within an EDITING phrase, an APPLY statement results in a GO action. The GO action is deferred until the end of the EDITING phrase.

This function is supported only for backward compatibility.

Note: Does not apply to SpeedScript programming.

Syntax

```
GO-PENDING
```

Example

The `r-gopend.p` procedure lets you update some of the fields in each customer record. If you press **GO** when the value in the current balance field is greater than the balance in the credit-limit field, the UPDATE statement does not end. Instead, it continues prompting you for input until you correct the problem and then press **GO**.

r-gopend.p

```
REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum.
  UPDATE
    Customer.Name Customer.Address Customer.City Customer.State SKIP
    Customer.CreditLimit Customer.Balance WITH 1 COLUMN EDITING:
    READKEY.
    APPLY LASTKEY.
    IF GO-PENDING AND INPUT Customer.Balance >
      INPUT Customer.CreditLimit THEN DO:
      MESSAGE "The current unpaid balance exceeds the credit limit.".
      NEXT.
    END.
  END. /* EDITING */
END.
```

See also

APPLY statement, EDITING phrase

GT or > operator

Returns a TRUE value if the first of two expressions is greater than the second expression.

Syntax

```
expression { GT | > } expression
```

expression

A constant, field name, variable name, or any combination of these. The expressions on either side of the GT or > must be of the same data type, although one might be integer and the other decimal.

Example

This procedure lists all items that have a negative `OnHand` quantity or more than 90% of the `OnHand` inventory currently allocated:

r-gt.p

```
FOR EACH Item NO-LOCK:
  IF Item.Allocated > 0 THEN IF (Item.OnHand <= 0) OR
    (Item.Allocated / Item.OnHand > .9) THEN
    DISPLAY Item.ItemNum Item.ItemName Item.OnHand Item.Allocated.
END.
```

Notes

- By default, the AVM uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (`-cpcoll`) startup parameter take precedence over a collation specified for any database the AVM accesses during the session, except when the AVM uses or modifies pre-existing indexes. If you do not specify a collation with the `-cpcoll` startup parameter, the AVM uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, the AVM uses the collation rules with the default name "`basic`" (which might or might not exist in the `convmap.cp` file).
- If either of the expressions is the Unknown value (?), then the result is the Unknown value (?); if both of the expressions are the Unknown value (?), then the result is FALSE.
- You can compare character strings with GT. Most character comparisons are case insensitive in ABL. That is, upper-case and lower-case characters have the same sort value. However, it is possible to define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If either *`expression`* is a field or variable defined as case sensitive, the comparison is case sensitive and "Smith" does not equal "smith".
- Characters are converted to their sort code values for comparison. Using the default case-sensitive collation table, all uppercase letters sort before all lowercase letters (for example, a is greater than Z, but less than b). Note also that in character code uppercase A is less than [, \ , ^ , _, and ` , but lowercase a is greater than these.

- You cannot compare data of different DATE, DATETIME, and DATETIME-TZ data types to each other using GT. You must first convert different date and datetime data types to the same data type before doing a comparison between them.
- You can use GT to compare a LONGCHAR variable to another LONGCHAR or CHARACTER variable. The variable values are converted to `-cpinternal` for comparison and must convert without error, or the AVM raises a run-time error.
- You cannot use GT to compare one CLOB field to another.
- You can use GT to compare two enums, including both ABL and .NET enums, as long as they are the same enum type. GT compares the underlying numerical values of the enumerators.

GUID function

Converts a universally unique identifier (UUID) value into a globally unique identifier (GUID) value. This function returns a GUID as a 36-character string value consisting of 32 hexadecimal digits (0 through 9 and A through F) and 4 hyphens formatted as follows (where X is a hexadecimal digit): XXXXXXXX–XXXX–XXXX–XXXX–XXXXXXXXXXXX.

Syntax

```
GUID [( UUID )]
```

UUID

An optional 16-byte raw UUID value to be converted. If the specified UUID is not exactly 16 bytes in length, the AVM returns the Unknown value (?). If not specified, the AVM generates a UUID and then converts it into a GUID.

Examples

The following code fragment illustrates how to use the GUID function:

```
DEFINE VARIABLE MyUUID AS RAW       NO-UNDO.
DEFINE VARIABLE vGUID  AS CHARACTER NO-UNDO.

ASSIGN
  MyUUID = GENERATE-UUID
  vGUID  = GUID(MyUUID).
```

The following code fragment generates a GUID value:

```
DEFINE VARIABLE vGUID  AS CHARACTER NO-UNDO.

ASSIGN
    vGUID  = GUID.
```

See also

GENERATE-UUID function

HANDLE function

Converts a string representation of a handle to a valid handle. If the string does not correlate to a valid handle in the session, the function returns 0.

Syntax

```
HANDLE ( handle-string )
```

Caution: Use this function only to convert a handle previously stored as a string value back to a valid handle. If you convert an arbitrary string to handle using this function and then reference the new handle, a system error will occur. If you use the VALID-HANDLE function to validate a handle generated from an arbitrary string value, a system error will occur.

handle-string

A string representation of a handle. Since handles are integer values, the string must contain only numeric characters.

Example

The following procedure creates a frame, stores the handle of the frame as a string value, deletes the frame, converts the string representation of the frame handle back to a valid handle, and then tests if the handle is valid:

r-widhd.p

```
DEFINE VARIABLE whand AS HANDLE    NO-UNDO.
DEFINE VARIABLE chand AS CHARACTER NO-UNDO.

CREATE FRAME whand.
chand = STRING(whand).
DELETE WIDGET whand.
whand = HANDLE(chand).
MESSAGE VALID-HANDLE(whand) VIEW-AS ALERT-BOX INFORMATION BUTTONS OK.
```

The VALID-HANDLE function returns a FALSE value because the frame was deleted and the handle is no longer valid.

Notes

- The HANDLE function can convert the string representation of procedure, system, and widget handles.
- For SpeedScript, the only valid use is to convert the handle of a QUERY object that you create using the CREATE WIDGET statement.

See also

CREATE widget statement, DATE function, DECIMAL function, INTEGER function, STRING function, VALID-HANDLE function

HEX-DECODE function

Converts a character string consisting of an even number of hexadecimal digits (0 through 9 and A through F) into a RAW value.

Syntax

```
HEX-DECODE( expression )
```

expression

A character expression containing the value you want to convert. If the expression does not contain an even number of hexadecimal digits, or it is the Unknown value (?), the result is the Unknown value (?). If the expression is a zero-length value, the result is a zero-length value.

Example

The following code fragment illustrates how to use the HEX-DECODE function:

```
DEFINE VARIABLE vRaw as RAW NO-UNDO.

vRaw = HEX-DECODE(HEX-ENCODE(GENERATE-UUID)).
```

See also

HEX-ENCODE function

HEX-ENCODE function

Converts a RAW value into a character string consisting of an even number of hexadecimal digits (0 through 9 and A through F).

Syntax

```
HEX-ENCODE( expression )
```

expression

A RAW expression containing the value you want to convert. If the expression is the Unknown value (?), the result is the Unknown value (?). If the expression is a zero-length value, the result is a zero-length value.

Example

The following code fragment illustrates how to use the HEX-ENCODE function:

```
DEFINE VARIABLE MyUUID AS RAW       NO-UNDO.
DEFINE VARIABLE vChar  AS CHARACTER NO-UNDO.

ASSIGN
  MyUUID = GENERATE-UUID
  vChar  = HEX-ENCODE(MyUUID).
```

See also

HEX-DECODE function

HIDE statement

Makes a widget invisible (sets its VISIBLE attribute to FALSE), or clears the message area for a window, or hides all widgets and clears messages in a window.

Note: Does not apply to SpeedScript programming.

Syntax

```
HIDE [ STREAM stream | STREAM-HANDLE handle ]
  [widget-phrase | MESSAGE | ALL ]
  [ NO-PAUSE ]
  [ IN WINDOW window ]
```

`STREAM` *`stream`*

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream.

`STREAM-HANDLE` *`handle`*

Specifies the handle to a stream. If *`handle`* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

`widget-phrase`

The widget you want to hide. You can hide windows, frames, and field-level widgets. You cannot hide menus. If you do not use this option or the MESSAGE or ALL options, HIDE hides the default frame for the block that contains the HIDE statement.

MESSAGE

Hides all messages displayed in the message area for the specified window. If you use the PUT SCREEN statement to display data in the message area, the HIDE MESSAGE statement does not necessarily hide that data.

ALL

Hides all widgets in the window and clears the message area for the window.

NO-PAUSE

Does not pause before hiding. Ordinarily, if data has been displayed, but there have been no data entry operations or pauses, the AVM prompts you to press **SPACEBAR** to continue before hiding the widget.

IN WINDOW *window*

Specifies which window the HIDE statement acts on. The value *window* must evaluate to the handle of a window. If you do not use the IN WINDOW option, the current window is assumed.

Example

The following example uses the HIDE statement to hide selected frames. The DISPLAY statements re-display the frames when the loop iterates.

r-hide.p

```
DEFINE VARIABLE selection AS INTEGER NO-UNDO FORMAT "9".

FORM
  "Please Make A Selection:" SKIP(2)
  "    1. Hide Frame A.    " SKIP
  "    2. Hide Frame B.    " SKIP
  "    3. Hide All.        " SKIP
  "    4. Hide This Frame  " SKIP
  "    5. Exit             " SKIP(2)
  WITH FRAME X NO-LABELS.

REPEAT:
  VIEW FRAME x.
  DISPLAY "This is frame A."
    WITH FRAME a ROW 1 COLUMN 60.
  DISPLAY "This is frame B."
    WITH FRAME b ROW 16 COLUMN 10 4 DOWN.
  MESSAGE "Make your selection!".
  UPDATE "Selection: " selection VALIDATE(0 < selection AND selection < 7,
    "Invalid selection") AUTO-RETURN
    WITH FRAME x.

       IF selection = 1 THEN HIDE FRAME a.
  ELSE IF selection = 2 THEN HIDE FRAME b.
  ELSE IF selection = 3 THEN HIDE ALL.
  ELSE IF selection = 4 THEN HIDE FRAME x.
  ELSE IF selection = 5 THEN LEAVE.
  PAUSE.
END.
```

Notes

- When a block iterates, any display frame that is scoped to the block or to a nested block is tagged for hiding (unless you have used the NO-HIDE option in the Frame phrase), but is not hidden. Then, the first frame activity of the next iteration (a DISPLAY, INSERT, PROMPT-FOR, SET, VIEW, or UPDATE statement) for a frame scoped to the block or to a nested block causes all tagged frames to be hidden.

  The frame associated with that first frame activity is not hidden because it would be redisplayed immediately. This improves display time. When a block ends, the AVM removes the hide tags from all the frames scoped to that block or to nested blocks.

- Frames displayed by procedures within a block or within a nested block are treated the same as other frames in a nested block.
- When the AVM displays a frame and there is not enough room in the window, it automatically hides one or more frames. Frames are hidden one at a time, starting with the lowest frame in the window, until there is room to fit the new frame.
- It is more efficient to HIDE ALL than to HIDE each frame individually.
- If you hide a PAGE-TOP or PAGE-BOTTOM frame, it is removed from the list of active frames for printing at the top or bottom of each page.
- If you are working in a PAGE-TOP or PAGE-BOTTOM frame, use the VIEW or DISPLAY statement to activate that frame. The VIEW statement does not display a PAGE-TOP or PAGE-BOTTOM frame. It activates the frame so that when a new page begins or ends, the AVM displays the frame. If you use the HIDE statement to hide a PAGE-TOP or PAGE-BOTTOM frame, the AVM deactivates that frame so that it can no longer be displayed unless it is reactivated with a VIEW or DISPLAY statement.
- If output is not directed to the terminal, HIDE has no effect on the terminal display.
- In batch mode, the HIDE statement produces an error. If you want to remove the contents of a frame, use the CLEAR statement instead.
- You can use HIDE MESSAGE to hide a message.
- If you invoke the HIDE statement for a field-level widget or child frame, the HIDDEN attribute of the specified field-level widget or child frame is also set to TRUE. However, if you invoke the HIDE statement for a child window, the HIDDEN attribute of the child window is unaffected.
- When you HIDE a visible window, any visible descendant windows are hidden also (including iconified descendants), but any visible ancestor windows remain unaffected. However, if you HIDE a window whose HIDDEN attribute is currently set to TRUE, its descendant windows remain unaffected.

See also

CLEAR statement, VIEW statement, Widget phrase, Stream object handle

---

IF...THEN...ELSE function

Evaluates and returns one of two expressions, depending on the value of a specified condition.

Syntax

```
IF condition THEN expression1 ELSE expression2
```

condition

An expression whose value is logical (TRUE or FALSE).

expression1

A constant, field name, variable name, or expression. If the *condition* is TRUE, then the function returns this value.

expression2

A constant, field name, variable name, or expression whose value is of a data type that is compatible with the data type of *expression1*. If the *condition* is FALSE or the Unknown value (?), then the function returns this value.

Example

You can use the IF . . . THEN . . . ELSE function when you want to sort records in an unusual order. In this example, the Customers are sorted so that those with a Balance greater than $10,000 appear first, then those with Balances between $1,000 and $10,000, and finally those with Balance of $1,000 or less:

r-ifelsf.p

```
FOR EACH Customer NO-LOCK BY IF Customer.Balance > 10000 THEN 1
  ELSE (IF Customer.Balance > 1000 THEN 2 ELSE 3) BY Customer.SalesRep:
  DISPLAY Customer.SalesRep Customer.Balance Customer.Name.
END.
```

IF...THEN...ELSE statement

Makes the execution of a statement or block of statements conditional. If the value of the expression following the IF statement is TRUE, the AVM processes the statements following the THEN statement. Otherwise, the AVM processes the statements following the ELSE statement.

Syntax

```
IF expression THEN { block | statement }
  [ ELSE { block | statement } ]
```

expression

A constant, field name, variable name, or expression whose value is logical (TRUE or FALSE). The expression can include comparisons, logical operators, and parentheses.

THEN

Describes the block statement to process if the *expression* is TRUE.

block

The block statement that contains the code you want to process if *expression* is TRUE. See the DO statement, FOR statement, and REPEAT statement reference entries for more information. If you do not start a block, you can process just one statement after the IF keyword or the ELSE keyword.

Any block or blocks you use in an IF...THEN...ELSE statement can contain other blocks or other IF...THEN...ELSE statements.

statement

A single ABL statement. The *statement* can be another IF...THEN...ELSE statement. If you want to use more than one statement, enclose those statements in a DO, FOR EACH, or REPEAT block.

ELSE

Describes the block statement to process if the *expression* is FALSE or the Unknown value (?). The ELSE option is not required.

Example

The r-ifelss.p procedure creates a report in a file that lists Customers whose Orders have been shipped, but who have not paid for those Orders:

r-ifelss.p

```
DEFINE VARIABLE ans AS LOGICAL NO-UNDO.

DEFINE STREAM due.

OUTPUT STREAM due TO ovrdue.lst.
DISPLAY STREAM due
  "Orders shipped but still unpaid as of" TODAY SKIP(2)
  WITH NO-BOX NO-LABELS CENTERED FRAME hdr PAGE-TOP.

FOR EACH Order WITH FRAME oinfo:
  FIND Customer OF Order NO-LOCK.
  DISPLAY Order.OrderNum Customer.Name Order.OrderDate Order.PromiseDate
    Order.ShipDate.
  IF Order.ShipDate = ? THEN DO:
    IF Order.PromiseDate = ? THEN DO:
      MESSAGE "Please update the promise date.".
      REPEAT WHILE promise-date = ?:
        UPDATE Order.PromiseDate WITH FRAME oinfo.
      END.
    END.
    ans = FALSE.
    MESSAGE "Has this order been shipped?" UPDATE ans.
    IF ans THEN
    REPEAT WHILE Order.ShipDate = ?:
      UPDATE Order.ShipDate WITH FRAME oinfo.
    END.
  END.
  ELSE DO:
    ans = TRUE.
    MESSAGE "Has this order been paid?" UPDATE ans.
    IF NOT ans THEN DO:
      DISPLAY STREAM due Order.OrderNum TO 14 Customer.Name AT 18
        Order.OrderDate AT 42 Order.ShipDate AT 54
        WITH NO-BOX DOWN FRAME unpaid.
    END.
  END.
END.
OUTPUT STREAM due CLOSE.
```

First, the procedure writes report headers to the `ovrdue.lst` file. Next, the outer `FOR EACH` block reads each of the Orders using a `DISPLAY` statement to display information on each Order. If there are no values in the `ShipDate` and `PromiseDate` fields, the procedure prompts you to enter a promise date. The procedure then prompts if the order has been shipped. If it has, supply a ship date.

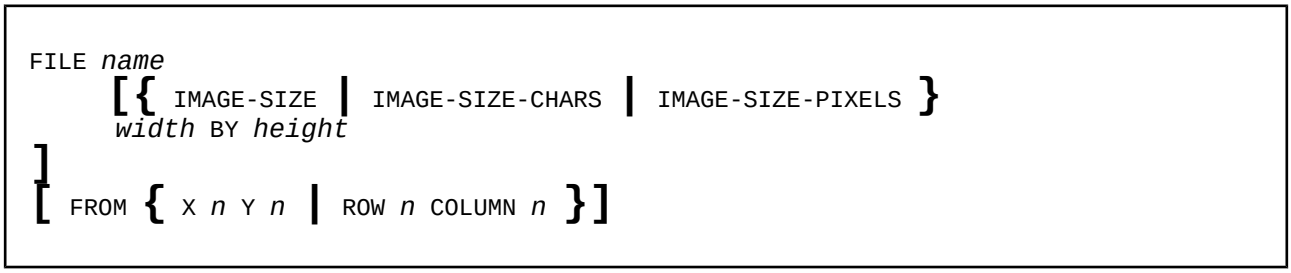
If there is a ship date and a promise date for an order, the procedure prompts if the order has been paid for. If not, the procedure displays the order information to the file.

Image phrase

Specifies the file in which an image is stored and the dimensions of the image.

Note: Does not apply to SpeedScript programming.

Syntax





`FILE` *`name`*

A character expression that specifies the name of an operating system file that contains an image. If you do not specify a full pathname, ABL searches your PROPATH for the file. If you do not supply a suffix, ABL searches for files with the extension `.bmp`, `.ico`, or `.cur` in Windows. The image contained within the file must be in a format that is appropriate for the target platform. The file is not read until the image is displayed. The filename can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

`[ IMAGE-SIZE | IMAGE-SIZE-CHARS ]`

Specifies that the unit of measure when reading the image is characters.

`IMAGE-SIZE-PIXELS`

Specifies that the unit of measure when reading the image is pixels.

`width`

Specifies the width of the image. The value *`width`* must be an integer constant. If the image is larger than the size you specify, the AVM crops the image to the specified size.

`height`

Specifies the height of the image. The value *`height`* must be an integer constant. If the image is larger than the size you specify, the AVM crops the image to the specified size.

`FROM { X` *`n`* `Y` *`n`* `| ROW` *`n`* `COL` *`n`* `}`

Two integer constants (*n*) that specify the offset inside the image file where the AVM starts reading the image. If you specify X and Y, the offset is measured in pixels; if you specify ROW and COL, the offset is measured in characters.

Example

See the DEFINE IMAGE statement reference entry for an example.

Notes

- Use one of the image size options in conjunction with the FILE option to make a compile-time association between the image file and the image widget; the image file does not have to exist at this point.
- Use one of the image size options without the FILE option to create an image widget that is not associated with an image file at compile time. You can then make the association at run time.
- Use the FILE option without one of the image size options if you do not know the size of the image and want ABL to determine the size at compile time. If you do this, ABL uses the entire image. Also note that the image file must exist or a compiler error will occur.
- In Windows, you can specify a URL pathname. If you do not specify a fully-qualified URL, ABL searches in the PROPATH for the file. Valid URL protocols include HTTP and HTTPS.

  Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, ABL continues searching with the next PROPATH entry.

- The following table lists the image file formats that are currently supported for use on button and image widgets.

  Table 43: Supported image file types

File extension	Image file type
.bmp	Windows bitmap
.gif **Note:** Animation in .gif files is not supported.	Graphics Interchange Format
.ico	Microsoft Icon File format
.jpg	JPEG
.png	GIF (Graphics Interchange Format) replacement
.tif	Tag image file format

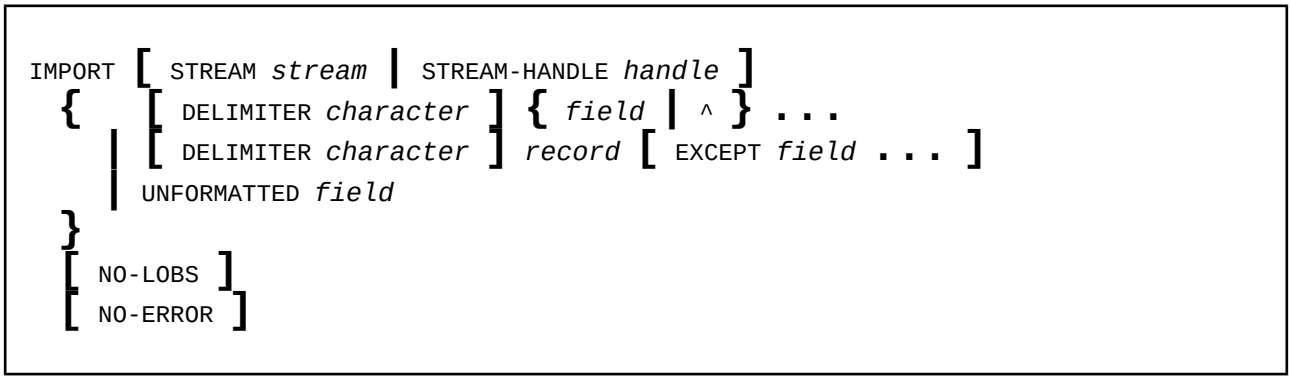
See also

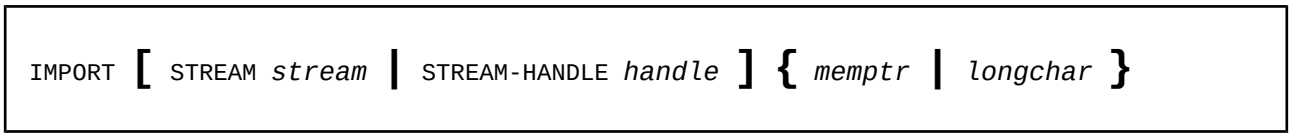
DEFINE BUTTON statement, DEFINE IMAGE statement, FORM statement

IMPORT statement

Reads a line from an input file that might have been created by EXPORT.

Syntax





`STREAM` *`stream`*

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream.

`STREAM-HANDLE` *`handle`*

Specifies the handle to a stream. If *`handle`* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

`DELIMITER` *`character`*

The character used as a delimiter between field values in the file. The *`character`* parameter must be a quoted **single** character. The default is a space character.

If you specify more than one character as a delimiter, ABL uses the first character as the delimiter.

`field`

The name of a field or variable to which you are importing data. The field or variable must have either the CHARACTER or RAW data type. If the data type is RAW, the `IMPORT` statement reads enough characters to fill the current length of the variable. If not enough characters are available to fill the current length, the length is reset to the number of characters read.

^

Use a caret (^) to skip a data value in each input line when input is being read from a file.

`record`

The name of a record buffer. All of the fields in the record are processed exactly as if you had named each of them individually. The record you name must contain at least one field. To use `IMPORT` with a record in a table defined for multiple databases, qualify the record's table name with the database name. See the *`record`* definition in the Record phrase reference entry for more information.

`EXCEPT` *`field`*

Tells the AVM to import all the fields except those listed in the `EXCEPT` phrase.

UNFORMATTED *field*

Treats each line of the input file as a single string value. In this case, the *field* parameter must be a single CHARACTER or RAW field or variable. You can use this option to read text files one line at a time.

Use this option on a RAW variable to import binary data that was not exported to the file as RAW data.

NO-LOBS

Directs the AVM to ignore large object data when importing records that contain BLOB or CLOB fields.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

memptr

A variable of data type MEMPTR to which you are importing data.

longchar

A variable of data type LONGCHAR to which you are importing data.

Examples

This procedure takes the data in file customer.d and enters it into the OpenEdge database table Customer. The procedure uses the DISABLE TRIGGERS statement to stop the AVM from executing any triggers for the CREATE, WRITE, and ASSIGN events when loading the data.

Note: The imported files, customer.d and custdump2, in the next two examples are created by running the example programs under EXPORT.

r-imprt.p

```
INPUT FROM customer.d.

DISABLE TRIGGERS FOR LOAD OF Customer.

REPEAT:
  CREATE Customer.
  IMPORT Customer.
END.

INPUT CLOSE.
```

If the file uses a delimiter other than a space to separate fields, use the DELIMITER option of the IMPORT statement.

r-cstin.p

```
DEFINE VARIABLE cnum  NO-UNDO LIKE Customer.CustNum.
DEFINE VARIABLE cname NO-UNDO LIKE Customer.Name.
DEFINE VARIABLE cmax  NO-UNDO LIKE Customer.CreditLimit.

INPUT FROM custdump2.

FOR EACH Customer:
  IMPORT DELIMITER ";" cnum cname cmax.
  DISPLAY cnum cname cmax.
END.

INPUT CLOSE.
```

You can use the UNFORMATTED option to read the contents of a standard text file. For example, the following procedure reads and displays the contents of the hello file:

r-hello.p

```
DEFINE VARIABLE text-string AS CHARACTER NO-UNDO FORMAT "x(76)".

INPUT FROM VALUE(SEARCH("hello")).
DO WHILE TRUE ON ENDKEY UNDO, LEAVE:
  IMPORT UNFORMATTED text-string.
  DISPLAY text-string WITH DOWN FRAME x.
  DOWN WITH FRAME x NO-LABELS.
END.
INPUT CLOSE.
```

In the MEMPTR version of the IMPORT statement, the MEMPTR must be pre-allocated to the size needed for reading. To get the length to read for an imported file, use the FILE_INFO system handle and the SET-SIZE statement as follows:

r-impmem.p

```
DEFINE VARIABLE bb AS MEMPTR NO-UNDO.

ASSIGN
  FILE-INFO:FILE-NAME = "big.in"
  SET-SIZE(bb)        = FILE-INFO:FILE-SIZE.

INPUT FROM "big.in" BINARY NO-CONVERT.
IMPORT bb.
INPUT CLOSE.
```

Notes

- The IMPORT statement must follow a statement that redirects the input source (usually an INPUT FROM statement). You cannot use the IMPORT statement to read data from the screen.
- If you do not use the UNFORMATTED option, the data in the input stream must be in a **standard format** to be read back into ABL. You must enclose all character fields in quotes ("") if they contain any delimiter characters. If you want to import any quotes contained in the data, replace them with two quotes ("" ""). You must display the Unknown value (?) as an unquoted question mark.

- If an input data line contains an unquoted hyphen in place of a data value, then the corresponding field is skipped, as it is in UPDATE. If you specify a hyphen (-) as the delimiter character, all hyphens are treated as delimiters. If you use the UNFORMATTED option, the hyphen is treated the same as any other character.
- A period (.) on a line by itself is treated as an end-of-file indicator. The ENDKEY is applied, but the file or stream remains open for input.
- Data read in with IMPORT is not restricted by frame-related format statements, as is data read in by SET or UPDATE. Since IMPORT does not have to validate the input stream, it is faster than SET or UPDATE.
- When importing records that contain a BLOB or CLOB field, the AVM uses the value stored in the BLOB or CLOB field of the exported record to determine whether or not the exported record has an associated object data file to import. If the BLOB or CLOB field in the exported record contains the Unknown value (?), the AVM stores the Unknown value (?) in the BLOB or CLOB field of the new or updated record. If the BLOB or CLOB field in the exported record contains a filename, the AVM imports the associated object data. If an updated record already has object data associated with it, the AVM deletes that object data before importing the new object data.

  The AVM raises the ERROR condition if an object data file cannot be found or read.
- Use the NO-LOBS option with the IMPORT statement to ignore large object data when importing records that contain BLOB or CLOB fields. More specifically:
  - When you import an exported record into a new record, and the BLOB or CLOB field of the exported record contains either the Unknown value (?) or a filename, the AVM sets the value of the BLOB or CLOB field in the newly imported record to the Unknown value (?); the AVM does not create any object data.
  - When you import an exported record as an update to an existing record, and the BLOB or CLOB field of the exported record contains either the Unknown value (?) or a filename, the AVM does not change the value of the BLOB or CLOB field in the existing record and neither creates nor overwrites object data.
- The IMPORT statement reads large object data files from the directory specified as the input data source in the INPUT FROM statement, by default. You can use the LOB-DIR option on the INPUT FROM statement to specify the directory from which the IMPORT statement reads BLOB and CLOB data files.
- IMPORT is sensitive to the Date Format (-d), Century (-yy), and European Numeric Format (-E) parameters. When loading data with the IMPORT statement, use the same settings that you used with the EXPORT statement.
- When importing DATETIME and DATETIME-TZ data, the data format must be fixed and must conform to the ISO 8601 standard for date/time representations (YYYY-MM-DDTHH:MM:SS.SSS+HH:MM). For DATETIME, there is no time zone offset.
- ABL interprets the null character as a terminator. If a complete record was not read before the null character was encountered, any buffer fields that were not read in will contain the values that were already in the buffer. Any subsequent records will not be read.
- The UNFORMATTED option forces IMPORT to read one physical line at a time. A physical line ends with a newline or linefeed control character, or the end of the file marker for the last line when it is not terminated with a new line or linefeed character.
- In the MEMPTR version of the IMPORT statement, the MEMPTR must be pre-allocated to the size needed for reading. See the example, r-impmem.p, above.
- When importing a CLOB field, the AVM uses the code page embedded in the filename of the <clobfile>.blb to determine the clob's code page. When importing a LONGCHAR variable, the AVM uses the code page information in the exported file header to determine the variable's code page. The COPY-LOB statement is recommended for Unicode CLOB fields or LONGCHAR variables, or when the code page information is not available in the source.

- When using the IMPORT statement to load data into an indeterminate array, all elements are loaded with the value of the first data value in the input record. To load each input data value into an element of the array, import the entire input file record into a CHARACTER variable and assign the individual values into the array elements in a loop.

See also

DEFINE STREAM statement, DISABLE TRIGGERS statement, DISPLAY statement, EXPORT statement, INPUT FROM statement, INPUT CLOSE statement, NO-ERROR option, PUT statement, Stream object handle, STRING function

INDEX function

Returns an INTEGER value that indicates the position of a search string within a source string.

Syntax

```
INDEX ( source , searchString [ , starting ] )
```

source

A CHARACTER, LONGCHAR or MEMPTR expression.

searchString

A CHARACTER or LONGCHAR expression whose position you want to locate in *source*. If *searchString* does not exist within *source*, INDEX returns 0.

starting

An integer that specifies at which left-most position in the string to start the search. For example, INDEX("abcdefabcdef","abc",6) returns 7.

The *starting* position is in characters if *source* is a CHARACTER or LONGCHAR expression. The *starting* position is in bytes if *source* is a MEMPTR.

Examples

For this example, you must enter 1, 2, 3, 4, or 5. The INDEX function checks if the digit exists in the string "12345".

r-index.p

```
DEFINE VARIABLE x    AS CHARACTER NO-UNDO FORMAT "9"
  LABEL "Enter a digit between 1 and 5".
DEFINE VARIABLE show AS CHARACTER NO-UNDO FORMAT "x(5)" EXTENT 5
  LABEL "Literal" INITIAL ["One", "Two", "Three", "Four", "Five"].

REPEAT:
  SET x AUTO-RETURN.
  IF INDEX("12345",x) = 0 THEN DO:
    MESSAGE "Digit must be 1,2,3,4, or 5. Try again.".
    UNDO, RETRY.
  END.
  ELSE DISPLAY show[INTEGER(x)].
END.
```

This procedure also uses the *starting* option:

r-index2.p

```
DEFINE VARIABLE positions AS CHARACTER NO-UNDO FORMAT "x(60)".
DEFINE VARIABLE sentence  AS CHARACTER NO-UNDO FORMAT "x(72)".
DEFINE VARIABLE vowel     AS CHARACTER NO-UNDO FORMAT "x".
DEFINE VARIABLE found     AS INTEGER   NO-UNDO.
DEFINE VARIABLE loop      AS INTEGER   NO-UNDO.
DEFINE VARIABLE start     AS INTEGER   NO-UNDO.

FORM sentence LABEL "Type in a sentence"
  WITH FRAME top
  TITLE "This program will tell where the vowels are in a sentence.".

SET sentence WITH FRAME top.
DO loop = 1 TO 5:
  ASSIGN
    positions = ""
    vowel     = SUBSTRING("aeiou",loop,1)
    start     = 1
    found     = INDEX(sentence,vowel,start).

  DO WHILE found > 0:
    ASSIGN
      positions = positions + STRING(found) + " "
      start     = found + 1
      found     = INDEX(sentence,vowel,start).
  END.
  DISPLAY vowel LABEL "Vowel" positions LABEL "Is found at locations..."
    WITH 5 DOWN.
  DOWN.
END.
```

The following example demonstrates using the INDEX function on a MEMPTR variable:

```
DEFINE VARIABLE mptr AS MEMPTR NO-UNDO.
DEFINE VARIABLE str AS CHAR NO-UNDO.
DEFINE VARIABLE str1 AS CHAR CASE-SENSITIVE NO-UNDO.

str = "Using MEMPTR as source param. MEMPTR is an ABL data type" .
SET-SIZE(mptr) = LENGTH(str, "raw") + 1.
PUT-STRING(mptr, 1) = str.
MESSAGE INDEX(mptr, "memptr", 10).  // returns 31

str1 = "memptr".
MESSAGE INDEX(mptr, str1, 10).  // returns 0
```

Notes

- If *source* is a CHARACTER or LONGCHAR expression and either operand is case sensitive, the search is case sensitive.
- If *source* is a MEMPTR, case sensitivity depends on the *searchString*. If *searchString* is case sensitive, the search is case sensitive.
- If *searchString* is an empty string ("") or a null string ("?"), the function returns 0.

See also

LOOKUP function, R-INDEX function, Using MEMPTR

INPUT function

References the value of a field in a frame. For example, if you use the PROMPT-FOR statement to get input from the user, PROMPT-FOR stores that information in the screen buffer. You can use the INPUT function to refer to that information.

Note: Does not apply to SpeedScript programming.

Syntax

```
INPUT [ FRAME frame ] field
```

FRAME *frame*

The name of the frame that contains the field named by the *field* argument. If you do not name a frame, the INPUT function starts with the current frame and searches outward until it finds the field you name with the *field* argument.

field

The name of a field or variable whose value is stored in the screen buffer. The specified field must be viewed as a fill-in or text widget.

Example

This procedure displays the current `CreditLimit` for a Customer. The PROMPT-FOR statement prompts the user for a new `CreditLimit` value and stores the supplied data in the screen buffer. The procedure uses the INPUT function to point to the data in that buffer.

r-input.p

```
FOR EACH Customer:
  DISPLAY Customer.CustNum Customer.Name Customer.CreditLimit
    LABEL "Current credit limit"
    WITH FRAME a 1 DOWN ROW 1.
  PROMPT-FOR Customer.CreditLimit LABEL "New credit limit"
    WITH SIDE-LABELS NO-BOX ROW 10 FRAME b.
  IF INPUT FRAME b Customer.CreditLimit <> Customer.CreditLimit THEN DO:
    DISPLAY "Changing max credit of" Customer.Name SKIP
      "from" Customer.CreditLimit "to" INPUT FRAME b Customer.CreditLimit
      WITH FRAME c ROW 15 NO-LABELS.
    Customer.CreditLimit = INPUT FRAME b Customer.CreditLimit.
  END.
  ELSE DISPLAY "No change in credit limit" WITH FRAME d ROW 15.
END.
```

If the user enters a new value, the procedure displays a message that the value has been changed. If the user enters the same value, the procedure displays a message that the credit-limit has not been changed.

Notes

- If you use a field or variable that is referenced with INPUT in more than one frame, then ABL uses the value in the frame most recently introduced in the procedure. To ensure that you are using the appropriate frame, use the FRAME option with the INPUT function to reference a particular frame.
- If you use the INPUT function for a character field whose format contains fill characters, then the value of the function does not contain the fill characters. The fill characters are not stored in the database field or variable, but are instead supplied during display formatting of the data.

INPUT CLEAR statement

Clears any keystrokes buffered from the keyboard, discarding any type-ahead characters. The INPUT CLEAR statement is useful when you want to make sure the AVM clears out extra characters in the input statement that could follow a field entry that is too long.

Note: Does not apply to SpeedScript programming.

Syntax

```
INPUT CLEAR
```

Example

This menu procedure tests each key the user presses. If the user presses a key other than 1, 2, or 3, the AVM clears the keyboard buffer and displays a message.

r-inclr.p

```
DISPLAY
  "     Please choose     " SKIP
  " 1  Run order entry    " SKIP
  " 2  Run receivables    " SKIP
  " 3  Exit               " WITH CENTERED FRAME menu.
REPEAT:
  READKEY.
  IF LASTKEY = KEYCODE("1") THEN RUN ordentry.
  ELSE
  IF LASTKEY = KEYCODE("2") THEN RUN receive.
  ELSE
  IF LASTKEY = KEYCODE("3") THEN QUIT.
  ELSE DO:
    MESSAGE "Sorry, that is not a valid choice".
    INPUT CLEAR.
  END.
END.
```

Notes

- In Windows, the keyboard type-ahead buffer can contain a maximum of 16 characters.
- If the current input source is not the keyboard, the INPUT CLEAR statement has no effect.
- INPUT CLEAR is available in Windows character and non-Windows GUI or character platforms.

See also

EDITING phrase

INPUT CLOSE statement

Closes the default input source or the stream you name.

Syntax

```
INPUT [ STREAM stream | STREAM-HANDLE handle ] CLOSE
```

`STREAM` *`stream`*

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream. See the DEFINE STREAM statement reference entry and *OpenEdge Programming Interfaces* for more information on streams.

`STREAM-HANDLE` *`handle`*

Specifies the handle to a stream. If *`handle`* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

Example

Instead of getting input from the terminal, the following procedure gets input from a file named `r-in.dat`. The SEARCH function determines the full pathname of this file.

r-in.p

```
INPUT FROM VALUE(SEARCH("r-in.dat")).

REPEAT:
  PROMPT-FOR Customer.CustNum Customer.CreditLimit.
  FIND Customer USING INPUT Customer.CustNum.
  ASSIGN Customer.CreditLimit.
END.

INPUT CLOSE.
```

Here is what the contents of the `r-in.dat` file looks like:

```
1 55800
2 41300
5 88000
```

The PROMPT-FOR statement uses the first data item (1) as the `CustNum` and the second data item (55800) as the `CreditLimit`. The FIND statement finds the Customer whose `CustNum` is 1 and assigns the value of 55800 as that Customer's credit limit. On the next iteration of the REPEAT block, the PROMPT-FOR statement uses the value of 2 as the `CustNum` the value of 41300 as the `CreditLimit`, etc.

The INPUT CLOSE statement closes the input source, resetting it to the terminal. When you run this procedure, the data in the window is simply an echo of the data as the procedure is reading it in from the `taxno.dat` file. If you do not want to display the data, add the word NO-ECHO to the end of the INPUT FROM statement.

Notes

- The default input source is the terminal unless the procedure was called by another procedure. In that case, the default input source is the one that was active in the calling procedure when the second procedure was called.
- When a procedure ends, the AVM closes all input sources established in that procedure.
- For more information on input sources, see *OpenEdge Programming Interfaces*.

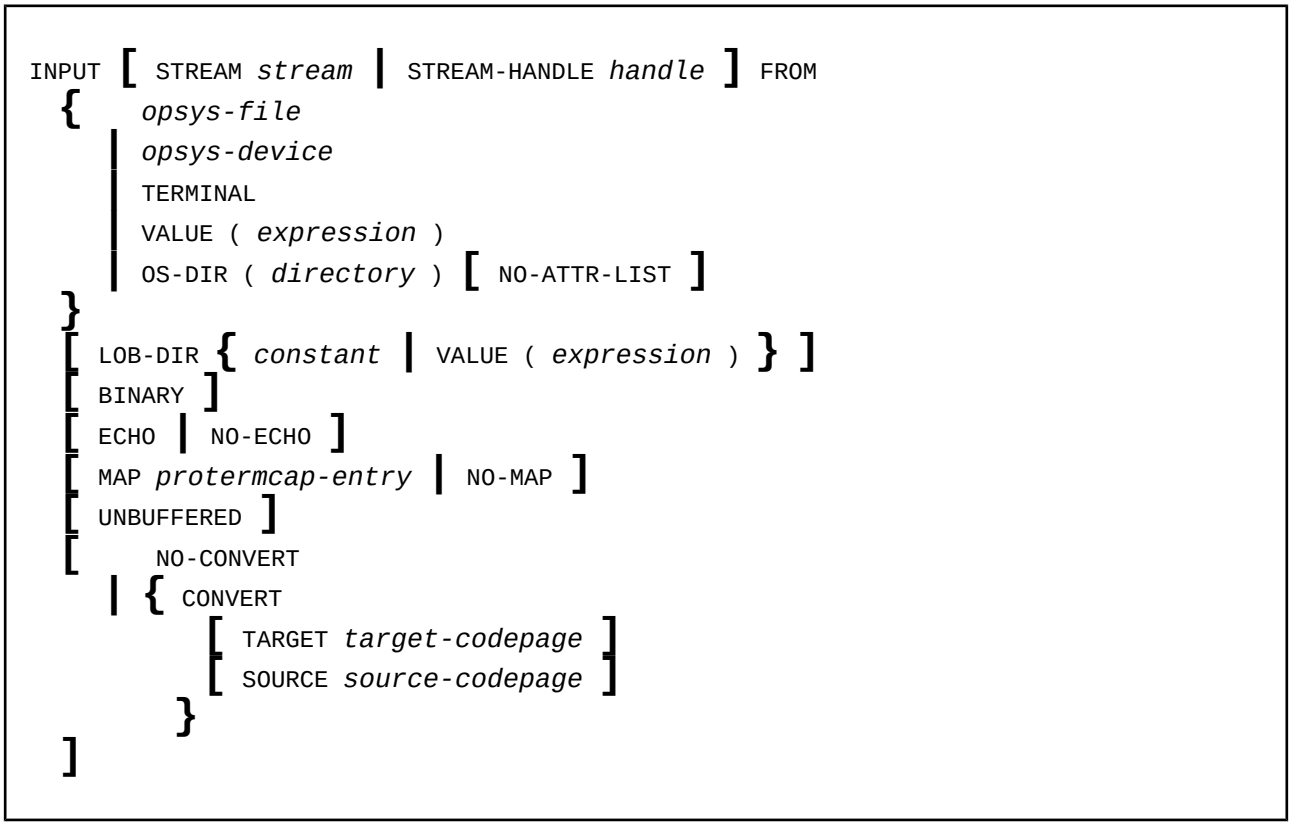
See also

DEFINE STREAM statement, INPUT FROM statement, Stream object handle

INPUT FROM statement

Specifies the new input source for a stream.

Syntax



STREAM *stream*

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream. See the DEFINE STREAM statement reference entry and *OpenEdge Programming Interfaces* for more information on streams.

STREAM-HANDLE *handle*

Specifies the handle to a stream. If *handle* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

opsys-file

The absolute or relative pathname of a file that contains the data you want to input. Any relative pathname is relative to the current working directory. Remember that UNIX file names are case sensitive. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

opsys-device

The name of a UNIX or Windows device.

TERMINAL

Indicates that you want to get input from your terminal. The terminal is the default input source. You cannot use TERMINAL with STREAM.

`VALUE ( expression )`

An expression whose value is the source where you want to input data. The value can contain Unicode characters.

`OS-DIR (directory)`

Indicates that you want your input to be the filenames found in *directory*. The value of *directory* is a character expression specifying an operating system directory. This value can contain Unicode characters. If *directory* is not a directory or you do not have permission to read it, then the INPUT statement raises ERROR. Otherwise, the AVM generates the directory list and feeds it back to the calling program through the INPUT stream. An INPUT CLOSE statement discards any unread filenames from the list.

Each line read from the input stream is a string composed of three tokens: the file's base name, the file's absolute path name, and an attribute list indicating the file type, which consists of one or more of the characters listed below.

You will get one of the following characters:

- **F** — Regular file or FIFO pipe
- **D** — Directory
- **S** — Special device
- **X** — Unknown file type

You might also get one or more of the following characters:

- **H** — Hidden file
- **L** — Symbolic link
- **P** — Pipe file

If you specify the NO-ATTR-LIST option, you will not get the attribute list for any line read from the input stream.

The two filenames in each input line are in EXPORT format; that is, they are enclosed in quotes and any embedded quotes are doubled. This means that INPUT FROM can process any filename, containing any characters, as long as IMPORT is used to read the input. The two filenames can use Unicode characters.

`NO-ATTR-LIST`

Omits the attribute list indicating the file type. This can speed up program execution. The following example illustrates this form of the statement:

```
INPUT FROM OS-DIR("c:\mydir") NO-ATTR-LIST.
```

`LOB-DIR { constant | VALUE ( expression ) }`

Specifies the directory from which you want the IMPORT statement to read large object data files (such as BLOB and CLOB data files). The *constant* and *expression* arguments are character expressions that evaluate to an absolute pathname or a relative pathname (relative to the directory specified for *opsys-file*).

If the specified character expression evaluates to either the Unknown value (?) or a directory that does not exist, or you do not have permission to read the specified directory, the AVM raises the ERROR condition.

The LOB-DIR option is valid only when you specify an operating system file as the input data source.

BINARY

Allows all input to be read directly without any conversion or interpretation. By default, NUL (\0) terminates character strings, and other control characters are interpreted as expected for the operating system.

ECHO

Displays all input data on the current output device. Data is echoed by default.

NO-ECHO

Accepts input data without displaying it on the current output device. If you do not use this option, INPUT FROM automatically displays input data on the current output device.

MAP *protermcap-entry* | NO-MAP

The *protermcap-entry* value is an entry from the PROTERMCAP file. Use MAP to read from an input stream that uses a different character translation from the current stream. Typically, *protermcap-entry* is a slash-separated combination of a standard device entry and one or more language-specific add-on entries (MAP laserwriter/french or MAP hp2/spanish/italian, for example). The AVM uses the PROTERMCAP entries to build a translation table for the stream. Use NO-MAP to make the AVM bypass character translation altogether. See *Internationalize ABL Applications* for more information on PROTERMCAP and national language support.

UNBUFFERED

Reads one character at a time from a normally buffered data source, such as a file. Use the UNBUFFERED option only when you can intermingle the input operations of a UNIX process, invoked with the ABL UNIX statement, with the input that follows the ABL INPUT FROM statement.

CONVERT

Allows you to modify the character conversions occurring between the external file and ABL. By default, the INPUT FROM statement converts characters from the code page specified with the Stream Code Page (-cpstream) parameter to the code page specified with the Internal Code Page (-cpinternal) parameter. If you specify SOURCE *source-codepage* alone, the conversion accepts *source-codepage* as the code page name of the external file (instead of -cpstream). If you specify TARGET *target-codepage*, the conversion accepts *target-codepage* as the internal code page (instead of -cpinternal). If you specify both SOURCE *source-codepage* and TARGET *target-codepage*, it converts characters from the *source-codepage* to *target-codepage* (instead of -cpstream to -cpinternal).

TARGET *target-codepage*

Specifies the target code page of the character conversion (replacing -cpinternal). The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that ABL uses for character management).

SOURCE *target-codepage*

Specifies the source code page of the character conversion (replacing -cpstream). The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that ABL uses for character management).

NO-CONVERT

Specifies that no character conversions occur between the external file and memory. By default, the INPUT FROM statement converts characters from the -cpstream code page to the -cpinternal code page.

Example

Instead of getting input from the terminal, this procedure gets input from a file named r-in.dat. The SEARCH function determines the full pathname of this file.

r-in.p

```
INPUT FROM VALUE(SEARCH("r-in.dat")).

REPEAT:
  PROMPT-FOR Customer.CustNum Customer.CreditLimit.
  FIND Customer USING INPUT Customer.CustNum.
  ASSIGN Customer.CreditLimit.
END.

INPUT CLOSE.
```

This is what the contents of the r-in.dat file look like:

```
1 55800
2 41300
5 88000
```

The PROMPT-FOR statement uses the first data item (1) as the CustNum and the second data item (55800) as the CreditLimit. The FIND statement finds the customer whose CustNum is 1 and assigns the value of 55800 as that Customer's credit limit. On the next iteration of the REPEAT block, the PROMPT-FOR statement uses the value of 2 as the CustNum, the value of 41300 as the CreditLimit, etc.

The INPUT CLOSE statement closes the input source, resetting it to the terminal. When you run this procedure, the data in the window is simply an echo of the data that the procedure is reading from the taxno.dat file. If you do not want to display the data, add the word NO-ECHO to the end of the INPUT FROM statement.

Notes

- To close the current input to a procedure, use the INPUT CLOSE statement. (The input source is automatically closed at the end of the procedure or when another default input source is opened.)
- The BINARY option allows you to use the READKEY statement to read control characters from the input source without interpretation. For example, NUL (\0) does not terminate strings, **CTRL+Z** does not signal EOF, and **CTRL+J** is not converted to **CTRL+M**, but their binary values are provided directly.
- If the input source and output destination are both the TERMINAL, then ECHO is always in effect.
- Use the IMPORT, INSERT, PROMPT-FOR, SET, or UPDATE statements to read data into an ABL procedure. The data is placed into the frame fields referenced in these statements, and, if you use ECHO, then the

frame is output to the current output destination. If you use the NO-ECHO option, then the frame is not output. If a subsequent DISPLAY statement causes the frame to appear, then the input data also appears if the frame is not yet in view.

- SEEK is not supported in conjunction with the OS-DIR option.
- When using the OS-DIR option, the UNBUFFERED option is ignored. OS-DIR always buffers exactly one filename at a time.
- If you use the PROMPT-FOR, SET, or UPDATE statement to read data from a file, the FORMAT for the data is ignored. Therefore, if you rely on FORMAT to validate input, you might read invalid characters.
- If you use the PROMPT-FOR, SET, or UPDATE statement to read data from a file, and there is a piece of data in each line of the file that you want to disregard, use a caret (^) in the PROMPT-FOR, SET, or UPDATE statement. For more information on this symbol, see the reference entry for any of those statements.
- If end of file is reached, the AVM responds as if you pressed **ENDKEY**.
- If a line consisting of a single period is read, that is treated as if you pressed **END-ERROR**. If the period is in quotes (".") it is treated as an ordinary character.
- When you use the INPUT FROM statement to read data from a file, there are two special characters you can use in that data file: tilde (~) and (slash (\) on UNIX, and hyphen (-).

  If characters in an input file take up more than one physical line, you can use tilde (~) to indicate a line continuation. This is an input file that uses a tilde:

```
92 "Match Point Tennis" "66 Homer Ave" "Como" ~
"TX" 75431
93 "Off the Wall" "20 Leedsville Ave" "Export" "PA" 15632
```

  Do not include a space after the tilde. For example:

Cust-num	Name	Address	City	State	Code
92	Match Point Tennis	66 Homer Ave	Como	TX	75431
93	Off The Wall	20 Leedsville Ave	Export	PA	15632

- You can see that the record containing the tilde was treated as a single input line.
- A hyphen in an input file indicates that you do not want to change the corresponding field in the INSERT, PROMPT-FOR, SET or UPDATE statement. This is the same input file as shown above, including the hyphen:

```
92 "Match Point Tennis" - "Como" "TX" 75431
93 "Off the Wall" "20 Leedsville Ave" "Export" "PA" 15632
```

  The procedure in the following example uses this file to set records in the customer file. When those records are displayed, the Match Point Tennis address does not change.

Cust-num	Name	Address	City	State	Code
92	Match Point Tennis		Como	TX	75431
93	Off The Wall	20 Leedsville Ave	Export	PA	15632

To enter a literal hyphen from a file, enclose it in quotes ("-").

- In Windows, the data in the input file must have the following characteristics:
  - The lines of data in the file are separated by CR-LF pairs.
  - There is no **CTRL+Z** (EOF) embedded in the file.
- For any character conversions to occur, all of the necessary conversion tables must appear in `convmap.cp` (a binary file that contains all of the tables that ABL uses for character management).
- If you specify a value of "undefined" for either *source-codepage* or *target-codepage*, no character conversion is performed.
- If the field being input is MEMPTR, you must use the BINARY and NO-CONVERT mode of operation to prevent your data from becoming corrupted if it contains binary data.
- With the BINARY and NO-CONVERT options, you will not get a translation of new-lines to the appropriate characters for your operating system and there will be no code page conversion between -cpinternal and -cpstream.
- If the field being input is MEMPTR and your MEMPTR contains ASCII data you may want code page conversion. However, you cannot get conversion by using the CONVERT parameter on the MEMPTR. You can get code page conversion by using the MEMPTR with the GET-STRING and CODEPAGE-CONVERT functions and the PUT-STRING statement.

See also

DEFINE STREAM statement, INPUT CLOSE statement, INPUT THROUGH statement, Stream object handle

INPUT THROUGH statement

Uses the output from a program as the input to an ABL procedure.

Syntax

```
INPUT [ STREAM stream | STREAM-HANDLE handle ] THROUGH
  { program-name | VALUE ( expression ) }
  [ argument | VALUE ( expression ) ] ...
  [ ECHO | NO-ECHO ]
  [ MAP protermcap-entry | NO-MAP ]
  [ UNBUFFERED ]
  [     NO-CONVERT
      | { CONVERT
            [ TARGET target-codepage ]
            [ SOURCE source-codepage ]
        }
  ]
```

`STREAM` *`stream`*

Specifies the name of a stream. If you do not name a stream, the unnamed stream is used. See the DEFINE STREAM statement reference entry and *OpenEdge Programming Interfaces* for more information on streams.

`STREAM-HANDLE` *`handle`*

Specifies the handle to a stream. If *`handle`* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

`program-name`

Represents the name of the program where you are supplying data to an ABL procedure. This can be a standard UNIX command or your own program. The name can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

`VALUE (` *`expression`*`)`

Specifies an expression whose value is the name of a program where you are supplying data to an ABL procedure. The name of the program can contain Unicode characters.

Or, it is an expression whose value is an argument you want to pass to the program. INPUT THROUGH passes the value of *`expression`* as a character string.

`argument`

Represents an argument you want to pass to the program. INPUT THROUGH passes this *`argument`* as a character string.

If the *`argument`* is the literal value echo, no-echo, or unbuffered, enclose it in quotes to prevent the AVM from interpreting that argument as one of the ECHO, NO-ECHO, or UNBUFFERED options for the INPUT THROUGH statement.

`ECHO`

Displays all input data on the current output destination. Data is echoed by default.

`NO-ECHO`

Accepts input data without displaying it on the current output device.

`MAP` *`protermcap-entry`* | `NO-MAP`

The *`protermcap-entry`* value is an entry from the PROTERMCAP file. Use MAP to read an input stream that uses a different character translation from the current stream. Typically, *`protermcap-entry`* is a slash-separated combination of a standard device entry and one or more language-specific add-on entries (MAP laserwriter/french or MAP hp2/spanish/italian, for example). The AVM uses the PROTERMCAP entries to build a translation table for the stream. Use NO-MAP to make the AVM bypass character translation altogether. See *Manage ABL Applications* for more information on PROTERMCAP. See *Internationalize ABL Applications* for more information on national language support.

UNBUFFERED

Reads one character at a time from a normally buffered data source, such as a file. Use the UNBUFFERED option only when the input operations of a UNIX process invoked by the ABL UNIX statement might be intermingled with the input from the ABL statements that follow the INPUT THROUGH statement.

CONVERT

Allows you to modify the character conversions occurring between the UNIX program and ABL. By default, the INPUT THROUGH statement converts characters from the code page specified with the Stream Code Page (-cpstream) parameter to the code page specified with the Internal Code Page (-cpinternal) parameter. If you specify SOURCE *source-codepage* alone, the conversion accepts *source-codepage* as the code page name of the UNIX program (instead of -cpstream). If you specify TARGET *target-codepage*, the conversion accepts *target-codepage* as the internal code page (instead of -cpinternal). If you specify both SOURCE *source-codepage* and TARGET *target-codepage*, it converts characters from the *source-codepage* to *target-codepage* (instead of -cpstream to -cpinternal).

TARGET *target-codepage*

Specifies the target code page of the character conversion (replacing -cpinternal). The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that ABL uses for character management).

SOURCE *target-codepage*

Specifies the source code page of the character conversion (replacing -cpstream). The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that ABL uses for character management).

NO-CONVERT

Specifies that no character conversions occur between the program and ABL. By default, the INPUT THROUGH statement converts characters from the -cpstream code page to the -cpinternal code page.

Examples

This procedure uses as its input source the output of the UNIX echo command. Before the command runs, the UNIX shell substitutes the process-id number for $$ and the current directory search path for $PATH. The results are then echoed and become available as a line of input to ABL. When the IMPORT statement is executed, the line of input from echo is read and the values are assigned to the two variables. Those variables can then be used for any purpose. In this example, the word echo must be lowercase and the word $PATH must be uppercase, since they both pass to UNIX:

r-ithru.p

```
DEFINE VARIABLE process-id AS CHARACTER NO-UNDO.
DEFINE VARIABLE dir-path   AS CHARACTER NO-UNDO
  VIEW-AS EDITOR SIZE 60 BY 10.

INPUT THROUGH echo $$ $PATH NO-ECHO.

SET process-id dir-path WITH FRAME indata NO-BOX NO-LABELS.
DISPLAY process-id dir-path FORMAT "x(70)".

INPUT CLOSE.
```

When you use INPUT THROUGH, the UNIX program you name is executed as a separate process under its own shell. Therefore, the values of shell variables (such as $$) are values from that shell rather than the shell from which the AVM executes.

The following procedure uses INPUT THROUGH twice to get input from the UNIX pwd and `ls` commands. The pwd command supplies the name of the current directory and the `ls` command supplies the name of each UNIX file in your current directory. After the variable `fn` is set, it displays on the screen.

r-ithru2.p

```
DEFINE VARIABLE dir-name AS CHARACTER NO-UNDO FORMAT "x(64)".
DEFINE VARIABLE fn       AS CHARACTER NO-UNDO FORMAT "x(32)".

FORM fn
  WITH DOWN FRAME dir-list.

/* Get the name of the current directory. */
INPUT THROUGH pwd NO-ECHO.
SET dir-name.
INPUT CLOSE.

/* Use the directory name as a label for fn. */
fn:LABEL IN FRAME dir-list = dir-name.

/* List the directory. */
INPUT THROUGH ls NO-ECHO.

/* For each entry in the directory, read the entry into fn and display it. */
REPEAT:
  SET fn WITH NO-BOX NO-LABELS FRAME indata.
  DISPLAY fn VIEW-AS TEXT WITH FRAME dir-list.
  DOWN WITH FRAME dir-list.
END.

INPUT CLOSE.
```

Notes

- INPUT THROUGH specifies the source for subsequent statements that process input. It does not read any data from the source.
- To use the IMPORT, INSERT, PROMPT-FOR, SET, or UPDATE statement, the AVM puts the data in the frame fields referenced in these statements, and if ECHO is in effect, the frame is output to the current output destination. If you use the NO-ECHO option, then the frame is not output. If a subsequent DISPLAY statement causes the frame to display, the input data also displays.
- When INPUT THROUGH is closed, the pipe to the UNIX process is also closed.
- For any character conversions to occur, all of the necessary conversion tables must appear in `convmap.cp` (a binary file that contains all of the tables that ABL uses for character management).
- If you specify a value of "undefined" for either *`source-codepage`* or *`target-codepage`*, no character conversion is performed.

See also

DEFINE STREAM statement, INPUT CLOSE statement, INPUT FROM statement, Stream object handle

INPUT-OUTPUT CLOSE statement

Closes a specified or default stream opened by an INPUT-OUTPUT THROUGH statement.

Syntax

```
INPUT-OUTPUT [ STREAM stream | STREAM-HANDLE handle ] CLOSE
```

STREAM *stream*

The name of the stream you want to close. If you do not name a stream, the AVM closes the default stream used by an INPUT-OUTPUT THROUGH statement.

STREAM-HANDLE *handle*

The handle to the stream you want to close. If you do not specify the handle, the AVM closes the default stream used by an INPUT-OUTPUT THROUGH statement.

Example

This procedure uses a C program to recalculate the price of each item in inventory. Specifically, the C program increases the price of each item by 3% or by 50 cents, whichever is greater. The INPUT-OUTPUT THROUGH statement tells the procedure to get its input from, and send its output to, the `r-iothru.p` procedure. The INPUT-OUTPUT CLOSE statement resets the input source to the terminal and the output destination to the terminal.

r-iothru.p

```
FOR EACH Item NO-LOCK WHERE Item.ItemNum < 10:
  DISPLAY Item.ItemNum Item.Price LABEL "Price before recalculation".
END.

INPUT-OUTPUT THROUGH r-iothru UNBUFFERED.

FOR EACH Item WHERE Item.ItemNum < 10:
  EXPORT Item.Price.
  SET Item.Price.
END.

INPUT-OUTPUT CLOSE.

FOR EACH Item WHERE Item.ItemNum < 10 WITH COLUMN 40:
  DISPLAY Item.ItemNum Item.Price LABEL "Price after recalculation".
END.
```

Note

For more information, see *OpenEdge Programming Interfaces*.

See also

DEFINE STREAM statement, INPUT-OUTPUT THROUGH statement, Stream object handle

INPUT-OUTPUT THROUGH statement

Names a program (process) for the AVM to start. This process is the input source as well as the output destination for the procedure.

Syntax

```
INPUT-OUTPUT [ STREAM stream | STREAM-HANDLE handle ]
  THROUGH { program-name | VALUE ( expression ) }
  [ argument | VALUE ( expression ) ] ...
  [ ECHO | NO-ECHO ]
  [ MAP protermcap-entry | NO-MAP ]
  [ UNBUFFERED ]
  [     NO-CONVERT
      | { CONVERT
            [ TARGET target-codepage ]
            [ SOURCE source-codepage]
        }
  ]
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, the unnamed stream is used. See the DEFINE STREAM statement reference entry and *OpenEdge Programming Interfaces* for more information on streams.

STREAM-HANDLE *handle*

Specifies the handle to a stream. If *handle* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

program-name

Identifies the name of the UNIX program where the procedure is getting data and where the procedure is sending data.

VALUE (*expression*)

Represents an expression whose value is the name of a UNIX program where the procedure is getting data and where the procedure is sending data.

Or, it is an expression whose value is an argument you want to pass to the UNIX program. INPUT-OUTPUT THROUGH passes the value of *expression* as a character string.

argument

Specifies an argument you want to pass to the UNIX program. INPUT-OUTPUT THROUGH passes this argument as a character string.

If the argument is the literal value echo, no-echo, or unbuffered, you must enclose it in quotes to prevent the AVM from interpreting that argument as one of the ECHO, NO-ECHO, or UNBUFFERED options for the INPUT-OUTPUT THROUGH statement.

ECHO

Displays all input data to the unnamed stream. Data is not echoed by default.

NO-ECHO

Accepts input data without displaying it on the current unnamed stream. Data is not echoed by default.

MAP *protermcap-entry* | NO-MAP

The *protermcap-entry* value is an entry from the PROTERMCAP file. MAP allows you to send output to and receive input from an I/O stream that uses different character translation than the current stream. Typically, *protermcap-entry* is a slash-separated combination of a standard device entry and one or more language-specific add-on entries (MAP laserwriter/french or MAP hp2/spanish/italian, for example). The AVM uses the PROTERMCAP entries to build a translation table for the stream. Use NO-MAP to make the AVM bypass character translation altogether. See *Manage ABL Applications* for more information on PROTERMCAP. See *Internationalize ABL Applications* for more information on national language support.

UNBUFFERED

Reads and writes one character at a time from a normally buffered data source, such as a file. Use the UNBUFFERED option only when the input-output operations of a process invoked by ABL's UNIX statement can be intermingled with the input-output from the ABL statements that follow the INPUT-OUTPUT THROUGH statement. INPUT-OUTPUT THROUGH handles the buffering of data between the ABL procedure and the UNIX program that it invokes. Use the UNBUFFERED option if your procedure invokes any other programs with the UNIX statement.

CONVERT

Allows you to modify the character conversions occurring between the UNIX program and ABL. By default, the INPUT-OUTPUT THROUGH statement converts characters from the Stream Code Page (-cpstream) parameter to the code page specified with the Internal Code Page (-cpinternal) parameter as data received from *program-name*. As data is passed to *program-name*, then INPUT-OUTPUT THROUGH converts from the -cpinternal to -cpstream. If you specify SOURCE *source-codepage* alone, the conversion accepts *source-codepage* as the code page name of the UNIX program (instead of -cpstream). If you specify TARGET *target-codepage*, the conversion accepts *target-codepage* as the internal code page (instead of -cpinternal). If you specify both SOURCE *source-codepage* and TARGET *target-codepage*, it converts characters from the *source-codepage* to *target-codepage* (instead of -cpstream to -cpinternal).

TARGET *target-codepage*

Specifies the target code page of the character conversion (replacing -cpinternal). The name that you specify must be a valid code page name available in the `DLC/convmap.cp` file (a binary file that contains all of the tables that ABL uses for character management).

SOURCE *target-codepage*

Specifies the source code page of the character conversion (replacing -cpstream). The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that ABL uses for character management).

NO-CONVERT

Specifies that no character conversions occur between the UNIX program and ABL. By default, the INPUT-OUTPUT THROUGH statement converts characters from the -cpstream code page to the -cpinternal code page as data is received from *program-name*. As data is passed to *program-name*, then INPUT-OUTPUT THROUGH converts from the -cpinternal to -cpstream.

Examples

This procedure uses a C program to recalculate the price of each item in inventory. Specifically, the C program increases the price of each item by 3% or by 50 cents, whichever is greater. The INPUT-OUTPUT THROUGH statement tells the procedure to get its input from, and send its output to, the r-iothru.p procedure. The INPUT-OUTPUT CLOSE statement resets the input source to the terminal and the output destination to the terminal.

r-iothru.p

```
FOR EACH Item WHERE Item.ItemNum < 10:
  DISPLAY Item.ItemNum Item.Price LABEL "Price before recalculation".
END.

INPUT-OUTPUT THROUGH r-iothru UNBUFFERED.

FOR EACH Item WHERE Item.ItemNum < 10:
  EXPORT Item.Price.
  SET Item.Price.
END.

INPUT-OUTPUT CLOSE.

FOR EACH Item WHERE Item.ItemNum < 10 WITH COLUMN 40:
  DISPLAY Item.ItemNum Item.Price LABEL "Price after recalculation".
END.
```

You can perform this calculation within a single ABL procedure. The C program is used for illustration purposes only. Use a UNIX program outside ABL to execute specialized calculations or processing.

You must unpack the C program from the proguide subdirectory and compile it before you can use it with the r-iothru.p procedure. If you do not have a C compiler, do not try this example.

Following is the C program used by the r-iothru.p procedure:

r-iothru.c

```
#include <stdio.h>
#define MAX(a,b)    ( (a < b) ? b : a )

main( )
 {
  float cost;

  /* This is important so that buffering does not   */
  /* defeat attempts to actually write on the pipe. */
  setbuf(stdout, (char *) NULL);


  while (scanf("%f", &cost) == 1) {
    /* Here the item cost is manipulated. We are */
    /* increasing the cost by a fixed percentage */
    /* (with a minimum increase), to provide     */
    /* an example.                               */
    cost = cost + MAX( 0.03 * cost, .50);
    printf("%10.2f\n", cost);
  }
}
```

Notes

- Use EXPORT or PUT, not DISPLAY, to write data to the program.
- Use SET to read data from the program.
- If you read data from a C program, put an upper limit on how many errors can occur before the program ends. Also remember that if the program prints an error message, that message is sent to ABL as data. You can use fprintf(stderr,...) to display debugging messages to the window, even in the middle of an INPUT-OUTPUT THROUGH operation.
- With INPUT-OUTPUT THROUGH in non-interactive mode, an ABL procedure can send information to a UNIX program, and the program can process that information and send the results back to the AVM. Some UNIX utilities you can use in batch mode are `wc` (word count) and `sort`.

  Here are some pointers for using INPUT-OUTPUT THROUGH in this way:

  - When the procedure finishes sending data to the program, use the OUTPUT CLOSE statement to reset the standard output stream to the screen. Doing this signals an EOF on the pipe, indicating that the program has received all input. When the procedure has received all data from the program, use the INPUT CLOSE statement to reset the standard input stream. Do not use the INPUT-OUTPUT CLOSE statement, because that closes both pipes at once.
  - If you want to use the INPUT-OUTPUT THROUGH statement with a UNIX utility that buffers its output, use the non-interactive approach.
  - To signal an EOF, use OUTPUT CLOSE (rather than attempting to send a **CTRL+D**).

  When you use INPUT-OUTPUT THROUGH in interactive mode the AVM sends data to the program, and the program sends data back to the AVM, etc.

  Here are some pointers for using INPUT-OUTPUT THROUGH in this way:

  - At the end of the interaction between the procedure and the program, use the INPUT-OUTPUT CLOSE statement to shut down both pipes.
  - Be sure that the program you are using does not buffer its output. If the program is a C program, the first line of the program should be "setbuf(stdout, (char *) NULL):". The program should also include "#include <stdio.h>". These tell UNIX that the standard output of the program is unbuffered. If the program

does buffer its output, use the batch approach to INPUT-OUTPUT THROUGH as explained in the previous note.

- If the program ends on some condition other than detecting an EOF, make sure that it tells the ABL procedure that it is about to end.

- For any character conversions to occur, all of the necessary conversion tables must appear in `convmap.cp` (a binary file that contains all of the tables that ABL uses for character management).
- If you specify a value of "undefined" for either *`source-codepage`* or *`target-codepage`*, no character conversion is performed.

See also

DEFINE STREAM statement, INPUT CLOSE statement, INPUT-OUTPUT CLOSE statement, Stream object handle

INSERT statement

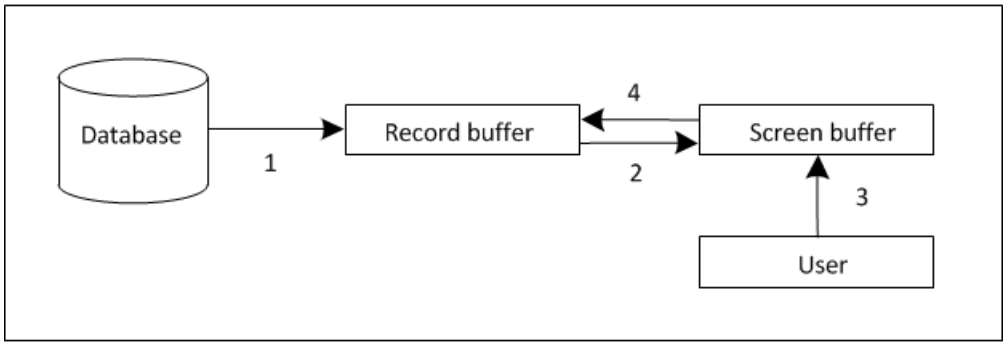
Creates a new database record, displays the initial values for the fields in the record, prompts for values of those fields, and assigns those values to the record.

The `INSERT` statement is a combination of the following statements:

- **CREATE** - Creates an empty record buffer
- **DISPLAY** - Moves the record from the record buffer into the screen buffer and displays the contents of the buffer on the screen
- **PROMPT-FOR** - Accepts input from the user, and puts that input into the screen buffer
- **ASSIGN** - Moves data from the screen buffer into the record buffer

Note: Does not apply to SpeedScript-programming.

Data movement



1. **CREATE** - Creates an empty record buffer
2. **DISPLAY** - Moves the contents of the record buffer to the screen buffer and displays the screen buffer
3. **PROMPT-FOR** - Accepts input from the user into the screen buffer
4. **ASSIGN** - Moves the contents of the screen buffer to the record buffer

Syntax

```
INSERT record [ EXCEPT field ... ]
  [ USING { ROWID ( nrow ) | RECID ( nrec ) } ]
  [ frame-phrase ]
  [ NO-ERROR ]
```

record

The name of the record you want to add to a database file. The AVM creates one record buffer for every file you use in a procedure. This buffer is used to hold a single record from the file associated with the buffer. Use the DEFINE BUFFER statement to create additional buffers, if necessary. The CREATE part of the INSERT statement creates an empty record buffer for the file in which you are inserting a record.

To insert a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

EXCEPT *field*

Inserts all fields except those listed in the EXCEPT phrase.

USING { ROWID (*nrow*) | RECID (*nrec*) }

Allows you to insert a record in an RMS relative file (for backward compatibility only) using a specific record number, where *nrow* is the ROWID relative record number of the record you want to insert and *nrec* is the RECID relative record number of the record you want to insert.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

In this procedure the user adds a new Order record. After the user adds a new Order record, the procedure creates OrderLines for that record. The procedure uses the CREATE statement to create OrderLines rather than the INSERT statement. When you use the INSERT statement, the PROMPT-FOR and ASSIGN parts of the INSERT let you put data into all the fields of the record being inserted. In the case of OrderLines, this procedure only lets you add information into a few of the OrderLine fields. Use CREATE together with UPDATE to single out the OrderLine fields.

r-insrt.p

```
  INSERT Order WITH 1 COLUMN.
  REPEAT:
    CREATE OrderLine.
    OrderLine.OrderNum = Order.OrderNum.
    UPDATE OrderLine.LineNum OrderLine.ItemNum OrderLine.Qty
      OrderLine.Price.
    /* Verify the ItemNum by finding an Item with that number */
    FIND Item OF OrderLine.
  END.
END.
```

Notes

- If an error occurs during the INSERT statement, the AVM retries the data entry part of the statement and processes the error associated with the block that contains the statement. (For example, an error might occur when the user enters a duplicate index value for a unique index.)REPEAT:
- When working with a partitioned table, creating a record without initially setting the partition fields to valid values may cause an error. See CREATE statement for more details.
- Any frame characteristics described by an INSERT statement contribute to the frame definition. When ABL compiles a procedure, it uses a top-to-bottom pass of the procedure to design all the frames required by that procedure, including those referenced by INSERT statements, and adds field and related format attributes as it goes through the procedure.
- If you receive input from a device other than the terminal, and the number of characters read by the INSERT statement for a particular field or variable exceeds the display format for that field or variable, the AVM returns an error. However, if you set a logical field that has a format of y/n and the data file contains a value of yes or no, the AVM converts that value to y or n.
- If you use a single qualified identifier with the INSERT statement, the compiler first interprets the reference as *dbname*.*filename*. If the compiler cannot resolve the reference as *dbname*.*filename*, it tries to resolve it as *filename*.*fieldname*.

  REPEAT:When inserting fields, you must use filenames that are different from field names to avoid ambiguous references. See the Record phrase reference entry for more information.
- The INSERT statement causes any related database CREATE triggers to execute. All CREATE triggers execute after the record is actually created. If a CREATE trigger fails (or executes a RETURN statement with the ERROR option), the record creation is undone.
- In the context of the .NET blocking method, System.Windows.Forms.Application:Run( ), if you execute the INSERT statement for a non-modal ABL window from within a user-defined function or non-VOID method, this statement raises the STOP condition. For more information on the .NET Application:Run( ) method, see the reference entry for the WAIT-FOR statement (.NET and ABL).

See also

DEFINE BUFFER statement, Frame phrase, NO-ERROR option

INT64 function

Takes any data type and returns an INT64 value, if the conversion is possible. This function takes most common data types except for RAW and MEMPTR.

Syntax

```
INT64 ( expression )
```

expression

A constant, field name, variable name, or expression whose value can be of any data type except for RAW and MEMPTR. The function behaves as described for the following data types:

- If the value of *expression* is a CHARACTER, it must be valid for conversion into a number (for example, "1.67" is valid, "1.x3" is not).
- If *expression* is an object reference (CLASS), the result is the ABL-generated ID for the class instance.
- If *expression* is a reference to an instance of an enum, the result is the underlying numeric value of that instance.
- If *expression* is a LOGICAL, the result is 0 if *expression* is FALSE and the result is 1 if *expression* is TRUE.
- If *expression* is a DATE, the result is the number of days from 1/1/4713 B.C. to that day. If *expression* is the Unknown value (?), the result is the Unknown value (?).

Example

The following example demonstrates how to use the INT64 function to convert a variety of data types to the INT64 data type:

r-int64.p

```
DEFINE VARIABLE myint   AS INTEGER   NO-UNDO
  INITIAL 2000000000 FORMAT "->>,>>>,>>>,>>>".
DEFINE VARIABLE myint64 AS INT64     NO-UNDO
  INITIAL 333444555666 FORMAT "->>,>>>,>>>,>>>,>>>".
DEFINE VARIABLE mychar  AS CHARACTER NO-UNDO INITIAL "2234.3".
DEFINE VARIABLE mydec   AS DECIMAL   NO-UNDO INITIAL "240,234.05".
DEFINE VARIABLE mybool  AS LOGICAL   NO-UNDO INITIAL TRUE.
DEFINE VARIABLE mydate  AS DATE      NO-UNDO INITIAL 01/01/06.
DEFINE VARIABLE myhdl   AS HANDLE    NO-UNDO.

DEFINE FRAME x WITH TITLE "emptyframe".
myhdl = FRAME x:handle.

MESSAGE "integer of integer expression is" INTEGER(myint).
MESSAGE "integer of int64 expression is" INTEGER(myint64 - 333000000000).
MESSAGE "integer of int64 >2gig is" INTEGER(myint64).
MESSAGE "integer of char is" INTEGER(mychar).
MESSAGE "integer of dec is" INTEGER(mydec).
MESSAGE "integer of bool is" INTEGER(mybool).
MESSAGE "integer of date is" INTEGER(mydate).
MESSAGE "integer of handle is" INTEGER(myhdl).
MESSAGE " ".

MESSAGE "int64 of integer expression is" INT64(myint).
MESSAGE "int64 of int64 expression is" INT64(myint64 - 333000000000).
MESSAGE "int64 of int64 >2gig is" INT64(myint64).
MESSAGE "int64 of char is" INT64(mychar).
MESSAGE "int64 of dec is" INT64(mydec).
MESSAGE "int64 of bool is" INT64(mybool).
MESSAGE "int64 of date is" INT64(mydate).
MESSAGE "int64 of handle is" INT64(myhdl).
MESSAGE " ".
```

See also

GET-INT64 function, PUT-INT64 statement, INTEGER function, DECIMAL function, STRING function

---

INTEGER function

Converts an expression of any data type, with the exception of BLOB, CLOB, and RAW, to a 32-bit integer value of data type INTEGER, rounding that value if necessary.

Syntax

```
INTEGER ( expression )
```

expression

A constant, field name, variable name, or expression. The function behaves as described for the following data types:

- If the value of *expression* is a CHARACTER, it must be valid for conversion into a number (for example, "1.67" is valid, "1.x3" is not).

- If *expression* is an object reference (CLASS), the result is the ABL-generated ID for the class instance.
- If *expression* is a reference to an instance of an enum, the result is the underlying numeric value of that instance. Because the underlying numeric values of enum members are of type INT64, the function may raise an error if the value cannot be represented as an INTEGER.
- If *expression* is a LOGICAL, the result is 0 if *expression* is FALSE and the result is 1 if *expression* is TRUE.
- If *expression* is a DATE, the result is the number of days from 1/1/4713 B.C. to that day. If *expression* is the Unknown value (?), the result is the Unknown value (?).

Example

This procedure takes the first word (that is, the substring that precedes the first space character) from the Customer `Address` and tries to convert it to an integer (street-number). If the conversion fails (for example, the first word contains non-numeric characters) the procedure displays an error message. Otherwise the `CustNum`, `Address`, and converted street number are displayed.

r-intgr.p

```
DEFINE VARIABLE street-number AS INTEGER NO-UNDO LABEL "Street Number".

FOR EACH Customer NO-LOCK:
  ASSIGN street-number = INTEGER(ENTRY(1, Customer.Address, " ")) NO-ERROR.

  IF ERROR-STATUS:ERROR THEN
    MESSAGE "Could not get street number of" Customer.Address.
  ELSE
    DISPLAY Customer.CustNum Customer.Address street-number.
END.
```

See also

DECIMAL function, STRING function

INTERFACE statement

Defines a user-defined interface. An interface defined with this statement represents a user-defined data type that defines a set of method, property, and event prototypes for methods, properties, and events that can be implemented by one or more classes. Any class that implements the interface must support all the methods, properties, and events whose prototypes are defined in the interface or any interface from which this interface inherits member prototypes.

You cannot instantiate an interface as an object. You can only use it to define the specified interface for a class (which you can instantiate) that implements the interface.

Note: This statement is applicable only when used in a class definition (`.cls`) file. For more information, see the Notes section in this reference entry.

Syntax

```
INTERFACE interface-type-name
[ INHERITS super-interface-name [ , super-interface-name ] ... ] :
interface-body
```

interface-type-name

Type-name syntax reference entry. You can also specify an unqualified interface name (without a package), but only if the interface name represents the full type name for the interface (that is, the interface is not defined as part of a package).

Note that you cannot specify "Progress" as the first component of the package name for any ABL user-defined interface. For example, "Progress.Inventory.IUpdateInv" is an invalid type name for a user-defined interface and results in a compiler error.The value of *interface-type-name* is restricted to alphanumeric characters plus the symbols #, $, %, and _.

INHERITS *super-interface-name* [, *super-interface-name*]

Optionally specifies the type name of one or more other interfaces (super interfaces) from which this interface inherits member prototypes as part of its definition. Each super-interface-name you specify can be an ABL or .NET interface type name as described in the Type-name syntax reference entry. With an appropriate USING statement, you can also specify an unqualified interface name (an interface name without the ABL package or .NET namespace that completes the interface type name).

Also note that if more than one occurrence of a super interface type or member prototype appears in the interface inheritance hierarchy, it is treated as a single occurrence. However, *interface-type-name* (the current interface definition type name) can never appear as any *super-interface-name*, and can never be inherited by any interface specified by a *super-interface-name*. In other words, there can be no cycles (recursive inheritance) in an interface inheritance hierarchy. Otherwise, ABL raises a compiler error.

interface-body

The body of an interface definition is composed of the following types of elements:

- Temp-table or ProDataSet object definitions used as parameters by one or more methods whose prototype is declared in this interface
- Method prototypes for common methods implemented by one or more classes
- Property prototypes for common properties implemented by one or more classes
- Event prototypes for common events implemented by one or more classes

Define elements in the interface body using the following syntax:

```
[ { temp-table | dataset } ... ]
[ method-prototypes ]
[ property-prototypes ]
[ event-prototypes ]
END [ INTERFACE ].
```

temp-table | *dataset*

Specifies one or more temp-table or ProDataSet object definitions used as parameters by one or more methods declared in this interface. You must specify these object definitions before any method prototypes. The AVM does not allocate memory for these object definitions. You cannot specify an access mode for these object definitions. These object definitions cannot be inherited from a super interface; they are private to the interface in which they are defined.

The definition of temp-table and ProDataSet object parameters for methods defined in any classes that implement this interface must match the temp-table or ProDataSet object definitions in this interface.

For temp-table objects:

- The temp-tables must have the same number of fields, and each field must match with respect to the data type, extent, and position. Neither the table names nor field names must match.
- The temp-tables must have the same number of indexes, and each index component must match, including the index names. However, the index-component field names do not need to match.

For ProDataSet objects:

- The ProDataSet objects must have the same number of member buffers, and the buffers must be in the same order. Neither the buffer names nor ProDataSet names must match.
- The temp-tables of these buffers must match as described above.

method-prototypes

Declares one or more method prototypes in the interface. A method prototype declares a method of a class without an implementation (that is, without specifying the method's logic or the END METHOD statement).

For information on the syntax for *method-prototype*, see the METHOD statement reference entry for declaring method prototypes. Note the following:

- The method access mode must be PUBLIC, either explicitly declared or defaulted.
- The method prototype must not include the STATIC, ABSTRACT, OVERRIDE, or FINAL options.

The implementation of these method prototypes, in classes that implement this interface, must not include the STATIC option and must match these declarations with respect to:

- Access mode, which must be PUBLIC
- Return type
- The number, data type, and access mode of any parameters

property-prototypes

Declares one or more property prototypes in the interface. A property prototype declares a property of a class without implementing its GET or SET accessors (that is, without specifying the property's logic). You must specify a property prototype with a PUBLIC access mode. The property prototype must include either a GET and SET accessor, or a GET accessor, or a SET accessor. Although the property declaration cannot have accessor implementations, the property when implemented in a class may include an accessor implementation.

For information on the syntax for *property-prototype*, see the DEFINE PROPERTY statement reference entry for declaring property prototypes. Note the following:

- The property access mode must be PUBLIC, either explicitly declared or defaulted.
- The property prototype must not include the STATIC, ABSTRACT, OVERRIDE, or INITIAL options.
- Accessors must not include access modifiers.
- Accessors must not include implementations.

The implementation of the property, in classes that implement this interface, must not include the STATIC option and must match these declarations with respect to:

- Access mode, which must be PUBLIC
- Name
- Data type or type name
- EXTENT — Its presence or absence, determinate or indeterminate, and size (if determinate)
- Accessors — Any accessor appearing in the interface, plus the option of any additional accessor not specified in the interface, all defaulting to PUBLIC access mode
- Presence or absence of NO-UNDO

event-prototypes

Declares one or more event prototypes in the interface. An event prototype declares a class event that the class must implement with an identical declaration, allowing the class to publish the event.

For information on the syntax for *event-prototype*, see the DEFINE EVENT statement reference entry for declaring class event prototypes. Note the following:

- The event access mode must be PUBLIC, either explicitly declared or defaulted.
- The event prototype must not include the STATIC, ABSTRACT, or OVERRIDE options.

The implementation of these event prototypes, in classes that implement this interface, must not include the STATIC option and must match these declarations with respect to:

- Access mode, which must be PUBLIC
- Any specified .NET delegate type or the number, data type, and access mode of any specified parameters

END [INTERFACE]

Specifies the end of the interface body definition. You must end the interface body definition with the END statement.

Examples

The following samples include two different class definitions that provide similar functionality, but in distinctly different ways. Each class implements the same interface and both classes define a ProDataSet data member (`dsHighCustData`) used to retrieve Defines the type name for a user-defined interface type. Specify an ABL interface type name as described in the `Customer` and related `Invoice` table data for a single `Customer` record from the `Sports2020` database. However, each class selects the `Customer` record using a different and functionally distinct algorithm provided in its own implementation of the same interface method prototype.

Thus, each of the following sample class files implements the `r-ICustObj` interface type defined in the class definition file, `r-ICustObj.cls` and provides the following functionality:

1. `r-ICustObjImpl.cls` — Defines the `r-ICustObjImpl` class, which retrieves data from the single `Customer` and related `Invoice` records that contain the highest balance value represented by the `Customer.Balance` value. This functionality is almost identical to what is provided by the `r-CustObj.cls` sample class file that is fully described in the Examples section of the CLASS statement reference entry. (The main differences from the `r-ICustObjImpl` class are that the `r-CustObj` class does not implement an interface type and it is defined as FINAL.)
2. `r-ICustObjImpl2.cls` — Defines the `r-ICustObjImpl2` class, which also retrieves data from a single `Customer` and related `Invoice` records. However, this class retrieves data for the `Customer` whose related `Invoice`Defines the type name for a user-defined interface type. Specify records contain the highest `Invoice` balance represented by the sum of their `Invoice.Amount` values.

Following is the interface definition provided by the sample `r-ICustObj.cls` file.

r-ICustObj.cls

```
INTERFACE r-ICustObj:

  /* Property prototypes to return basic values for the Customer
     identified with the highest balance in the database */
  DEFINE PUBLIC PROPERTY HighCustBalance AS DECIMAL NO-UNDO
    GET.
    SET.
  DEFINE PUBLIC PROPERTY HighCustNum AS INTEGER NO-UNDO
    GET.
    SET.

  /* Event prototype to notify about Customers with Invoices */
  DEFINE PUBLIC EVENT CustHasInvoices
    SIGNATURE VOID ( piCustNum AS INTEGER ).

  /* Temp-tables for the ProDataSet parameter */
  DEFINE TEMP-TABLE ttCust NO-UNDO LIKE Customer.
  DEFINE TEMP-TABLE ttInv  NO-UNDO LIKE Invoice.

  /* ProDataSet parameter for passing a single Customer with the highest
     balance along with its related Invoices */
  DEFINE DATASET dsHighCustData FOR ttCust, ttInv
    DATA-RELATION FOR ttCust, ttInv
      RELATION-FIELDS (ttCust.CustNum, ttInv.CustNum).

  /* Method prototype to get the current high Customer balance data */
  METHOD PUBLIC VOID GetHighCustomerData
    ( OUTPUT DATASET dsHighCustData BIND ).

  /* Method prototype to set (or reset) the current high Customer data */
  METHOD PUBLIC VOID SetHighCustomerData ( ).

END INTERFACE.
```

The implementations of the `SetHighCustomerData( )` method populate the ProDataSet with selected fields from `Customer` and `Invoice` records, where the selected `Customer` also has related invoices. Each implementation of `SetHighCustomerData( )` also sets the two implemented properties (`HighCustBalance` and `HighCustNum`) to appropriate values for the selected `Customer`, and publishes the implemented class event (`CustHasInvoices`) for each `Customer` record it encounters with related `Invoice` records. Each class defines additional data members to support its instance of the ProDataSet, and also must implement the `GetHighCustomerData( )` method (according to the interface) to pass the ProDataSet as a by-reference output parameter.

Following is the interface implementation provided by the `r-ICustObjImpl` sample class. Note that the property implementations add initial values, but rely on default behavior for the accessors. The class also fully defines the ProDataSet to be passed as a by-reference method output parameter.

r-ICustObjImpl.cls

```
CLASS r-ICustObjImpl IMPLEMENTS r-ICustObj:

  /* Public properties to return basic values for a customer with the
     highest balance */
  DEFINE PUBLIC PROPERTY HighCustBalance AS DECIMAL INITIAL 0.0 NO-UNDO
    GET.
    SET.
  DEFINE PUBLIC PROPERTY HighCustNum AS INTEGER INITIAL ? NO-UNDO
    GET.
    SET.

  /* Public event to notify about Customers with Invoices */
  DEFINE PUBLIC EVENT CustHasInvoices
    SIGNATURE VOID ( piCustNum AS INTEGER ).

  /* Private handle variable for the high customer ProDataSet */
  DEFINE PRIVATE VARIABLE hHighCustData AS HANDLE NO-UNDO.

  /* Private temp-tables for the high customer ProDataSet */
  DEFINE PRIVATE TEMP-TABLE ttCust NO-UNDO LIKE Customer.
  DEFINE PRIVATE TEMP-TABLE ttInv  NO-UNDO LIKE Invoice.

  /* Private ProDataSet for a single customer with the highest
     balance and its invoices */
  DEFINE PRIVATE DATASET dsHighCustData FOR ttCust, ttInv
    DATA-RELATION FOR ttCust, ttInv
      RELATION-FIELDS (ttCust.CustNum, ttInv.CustNum).

  /* Private query and data sources for the ProDataSet */
  DEFINE PRIVATE QUERY qCust FOR Customer.
  DEFINE PRIVATE DATA-SOURCE srcCust FOR QUERY qCust.
  DEFINE PRIVATE DATA-SOURCE srcInv  FOR Invoice.

  /* Constructor to initialize handles and attach data sources */
  CONSTRUCTOR r-ICustObjImpl ( ):

    hHighCustData = DATASET dsHighCustData:HANDLE.
    BUFFER ttCust:ATTACH-DATA-SOURCE( DATA-SOURCE srcCust:HANDLE ).
    BUFFER ttInv:ATTACH-DATA-SOURCE( DATA-SOURCE srcInv:HANDLE ).

  END CONSTRUCTOR.

/* Public method to get the current high customer data */
  METHOD PUBLIC VOID GetHighCustomerData
    ( OUTPUT DATASET dsHighCustData BIND ):

  END METHOD.

  /* Public method to set (or reset) the current high customer data */
  METHOD PUBLIC VOID SetHighCustomerData ( ):

    hHighCustData:EMPTY-DATASET( ).
    FOR EACH Customer: /* Find Customer with highest balance */
      FIND FIRST Invoice WHERE Invoice.CustNum = Customer.CustNum NO-ERROR.
      IF AVAILABLE Invoice THEN DO:
        IF Customer.Balance > HighCustBalance THEN
          ASSIGN  HighCustBalance = Customer.Balance
                  HighCustNum     = Customer.CustNum.
        CustHasInvoices:Publish( Customer.CustNum ).
      END.
    END.
    QUERY qCust:QUERY-PREPARE("FOR EACH Customer "
      + "WHERE Customer.CustNum = " + STRING(HighCustNum) ).
    hHighCustData:FILL( ).

  END METHOD.
```

```
END CLASS.
```

The bold code inside the `SetHighCustomerData( )` method shows the difference in implementation from the same method implemented by the following `r-ICustObjImpl2` sample class. Here, it relies on the stored value of the `Customer.Balance` field to determine each `Customer` balance and assigns the `HighCustBalance` and `HighCustNum` property values accordingly.

The following `r-ICustObjProc.p` sample procedure shows an application of the `r-ICustObjImpl` class, which responds to the `CustHasInvoices` class event and displays the contents of the `dsHighCustCata` ProDataSet. Note that it defines a reference-only instance of the ProDataSet as required by the class and its interface. (This application is identical to the `r-CustObjProc.p` sample procedure described in the Examples section of the CLASS statement, but which instantiates the `r-CustObj` sample class instead.)

r-ICustObjProc.p

```
DEFINE TEMP-TABLE ttCust NO-UNDO REFERENCE-ONLY LIKE Customer.
DEFINE TEMP-TABLE ttInv  NO-UNDO REFERENCE-ONLY LIKE Invoice.

DEFINE DATASET dsHighCustDataREFERENCE-ONLY FOR ttCust, ttInv
  DATA-RELATION FOR ttCust, ttInv
    RELATION-FIELDS (ttCust.CustNum, ttInv.CustNum).

DEFINE VARIABLE rObj AS CLASS r-ICustObjImpl NO-UNDO.

rObj = NEW r-ICustObjImpl( ) NO-ERROR.
rObj:CustHasInvoices:Subscribe( "CustHasInvoices_Handler" ) NO-ERROR.

MESSAGE "High Customer Number:" rObj:HighCustNum SKIP
        "High Customer Balance:" rObj:HighCustBalance VIEW-AS ALERT-BOX.

rObj:SetHighCustomerData( ) NO-ERROR.

MESSAGE "High Customer Number:" rObj:HighCustNum SKIP
        "High Customer Balance:" rObj:HighCustBalance VIEW-AS ALERT-BOX.

rObj:GetHighCustomerData( OUTPUT DATASET dsHighCustData BIND ) NO-ERROR.

CURRENT-WINDOW:WIDTH-CHARS = 90.

FOR EACH ttCust, EACH ttInv BREAK BY ttInv.CustNum:
  DISPLAY ttCust.CustNum WHEN FIRST-OF(ttInv.CustNum)
          ttCust.Name WHEN FIRST-OF(ttInv.CustNum)
          ttCust.Balance WHEN FIRST-OF(ttInv.CustNum)
          ttInv.InvoiceNum ttInv.Amount SKIP
    WITH FRAME A WIDTH 90 DOWN
         TITLE "Customer with highest stored balance" NO-ERROR.
END.

PROCEDURE CustHasInvoices_Handler:
  DEFINE INPUT PARAMETER pCustNum AS INTEGER.

  FIND FIRST Customer WHERE Customer.CustNum = pCustNum NO-ERROR.
  IF AVAILABLE Customer THEN
    MESSAGE "Customer" Customer.CustNum ('"' + Customer.Name + '"')
            "has a stored balance of" Customer.Balance
            "and also has Invoices."
            VIEW-AS ALERT-BOX.

END PROCEDURE.
```

Following is the interface implementation provided by the `r-ICustObjImpl2` sample class. This implementation is identical to `r-ICustObjImpl` except for the `SetHighCustomerData( )` method.

r-ICustObjImpl2.cls

```
CLASS r-ICustObjImpl2 IMPLEMENTS r-ICustObj:

  /* Public properties to return basic values for a customer with the
     highest balance */
  DEFINE PUBLIC PROPERTY HighCustBalance AS DECIMAL INITIAL 0.0 NO-UNDO
    GET.
    SET.
  DEFINE PUBLIC PROPERTY HighCustNum AS INTEGER INITIAL ? NO-UNDO
    GET.
    SET.

  /* Public event to notify about Customers with Invoices */
  DEFINE PUBLIC EVENT CustHasInvoices
    SIGNATURE VOID ( piCustNum AS INTEGER ).

  /* Private handle variable for the high customer ProDataSet */
  DEFINE PRIVATE VARIABLE hHighCustData AS HANDLE NO-UNDO.

  /* Private temp-tables for the high customer ProDataSet */
  DEFINE PRIVATE TEMP-TABLE ttCust NO-UNDO LIKE Customer.
  DEFINE PRIVATE TEMP-TABLE ttInv  NO-UNDO LIKE Invoice.

  /* Private ProDataSet for a single customer with the highest
     balance and its invoices */
  DEFINE PRIVATE DATASET dsHighCustData FOR ttCust, ttInv
    DATA-RELATION FOR ttCust, ttInv
      RELATION-FIELDS (ttCust.CustNum, ttInv.CustNum).

  /* Private query and data sources for the ProDataSet */
  DEFINE PRIVATE QUERY qCust FOR Customer.
  DEFINE PRIVATE DATA-SOURCE srcCust FOR QUERY qCust.
  DEFINE PRIVATE DATA-SOURCE srcInv  FOR Invoice.

  /* Constructor to initialize handles and attach data sources */
  CONSTRUCTOR r-ICustObjImpl2 ( ):

    hHighCustData = DATASET dsHighCustData:HANDLE.
    BUFFER ttCust:ATTACH-DATA-SOURCE( DATA-SOURCE srcCust:HANDLE ).
    BUFFER ttInv:ATTACH-DATA-SOURCE( DATA-SOURCE srcInv:HANDLE ).

  END CONSTRUCTOR.

 /* Public method to get the current high customer data */
  METHOD PUBLIC VOID GetHighCustomerData
    ( OUTPUT DATASET dsHighCustData BIND ):

  END METHOD.

  /* Public method to set (or reset) the current high customer data */
  METHOD PUBLIC VOID SetHighCustomerData ( ):

    DEFINE VARIABLE dBalance AS DECIMAL NO-UNDO.

    hHighCustData:EMPTY-DATASET( ).
    /* Find Customer with highest total Invoice balance */
    FOR EACH Customer:
      ASSIGN dBalance = 0.0.
      FOR EACH Invoice WHERE Invoice.CustNum = Customer.CustNum:
        ASSIGN dBalance = dBalance + Invoice.Amount.
      END.
      IF dBalance > HighCustBalance THEN
        ASSIGN  HighCustBalance = dBalance
                HighCustNum     = Customer.CustNum.
```

```
      IF dBalance > 0.0 THEN
        CustHasInvoices:Publish( Customer.CustNum ).
    END.
    QUERY qCust:QUERY-PREPARE("FOR EACH Customer "
      + "WHERE Customer.CustNum = " + STRING(HighCustNum) ).
    hHighCustData:FILL( ).

  END METHOD.

END CLASS.
```

The bold code inside the `SetHighCustomerData( )` method shows the difference in implementation from the same method implemented by the previous `r-ICustObjImpl` sample class. Here, it calculates the `Customer` balance from the total of `Invoice.Amount` values in its related `Invoice` records and assigns the `HighCustBalance` and `HighCustNum` property values accordingly.

The following `r-ICustObjProc2.p` sample procedure shows an application of the `r-ICustObjImpl2` class, which is very similar to the previous procedure, `r-ICustObjProc.p`. The differences include displaying both the stored `Customer.Balance` value and the `Invoice.Amount` total for the selected `Customer` record, as well as some cosmetic changes to the display. Otherwise, the application is identical.

r-ICustObjProc2.p

```
DEFINE TEMP-TABLE ttCust NO-UNDO REFERENCE-ONLY LIKE Customer.
DEFINE TEMP-TABLE ttInv  NO-UNDO REFERENCE-ONLY LIKE Invoice.

DEFINE DATASET dsHighCustData REFERENCE-ONLY FOR ttCust, ttInv
  DATA-RELATION FOR ttCust, ttInv
    RELATION-FIELDS (ttCust.CustNum, ttInv.CustNum).

DEFINE VARIABLE rObj AS CLASS r-ICustObjImpl2 NO-UNDO.

rObj = NEW r-ICustObjImpl2( ) NO-ERROR.
rObj:CustHasInvoices:Subscribe( "CustHasInvoices_Handler" ) NO-ERROR.

MESSAGE "High Customer Number:" rObj:HighCustNum SKIP
        "High Invoice Balance:" rObj:HighCustBalance VIEW-AS ALERT-BOX.

rObj:SetHighCustomerData( ) NO-ERROR.

MESSAGE "High Customer Number:" rObj:HighCustNum SKIP
        "High Invoice Balance:" rObj:HighCustBalance VIEW-AS ALERT-BOX.

rObj:GetHighCustomerData( OUTPUT DATASET dsHighCustData BIND ) NO-ERROR.

CURRENT-WINDOW:WIDTH-CHARS = 100.

FOR EACH ttCust, EACH ttInv BREAK BY ttInv.CustNum:
  DISPLAY ttCust.CustNum WHEN FIRST-OF(ttInv.CustNum)
            COLUMN-LABEL "Customer!Number"
          ttCust.Name WHEN FIRST-OF(ttInv.CustNum)
          ttCust.Balance WHEN FIRST-OF(ttInv.CustNum)
            COLUMN-LABEL "Stored!Balance"
          ttInv.InvoiceNum COLUMN-LABEL "Invoice!Number"
          ttInv.Amount (SUB-TOTAL BY ttInv.CustNum) SKIP
    WITH FRAME A WIDTH 100 DOWN
         TITLE "Customer with highest total Invoice balance" NO-ERROR.
END.

PROCEDURE CustHasInvoices_Handler:
  DEFINE INPUT PARAMETER pCustNum AS INTEGER.

  FIND FIRST Customer WHERE Customer.CustNum = pCustNum NO-ERROR.
  IF AVAILABLE Customer THEN
```

```
    MESSAGE "Customer" Customer.CustNum ('"' + Customer.Name + '"')
            "has a stored balance of" Customer.Balance
            "and also has Invoices."
            VIEW-AS ALERT-BOX.

END PROCEDURE.
```

Notes

- You can terminate an INTERFACE statement with either a period (.) or a colon (:), but typically use a colon (:).
- A class definition (`.cls`) file can contain only one interface definition that is optionally preceded by one or more USING statements. The complete interface definition must begin with the INTERFACE statement and end with the END statement, and the INTERFACE statement must be the first compilable statement after any USING statements in the file. A class definition file containing an interface definition cannot also contain a class definition.
- The access mode for an interface definition is always PUBLIC.
- The compiled version of an interface definition file is an r-code (`.r`) file. For more information, see the COMPILE statement reference entry.
- You can define an object reference variable for an interface, which lets you reference a class that implements the interface, but you **cannot** create an instance of an interface with the NEW function (classes). For more information on object references, see the AS CLASS option in the DEFINE VARIABLE statement reference entry and see the Class-based object reference reference entry.
- You can reference include files from within an interface definition. For more information about include files, see the { } Include file reference entry.
- All built-in preprocessor directives are supported in interface definitions.
- All built-in preprocessor names are supported in interface definitions. For a list of preprocessor name, see the { } Preprocessor name reference entry.
- You cannot pass compile-time arguments to interface definition files. However, you can pass compile-time arguments to include files referenced in an interface definition file.
- You can store class definition r-code files in ABL procedure libraries. If ABL encounters a procedure library on PROPATH, it will search the library for the specified r-code. However, you cannot execute r-code files stored in a procedure library that is not on PROPATH using the *procedure-library-path*<<*member-name*>> syntax.
- While a property interface declaration may include a GET and SET accessor, or a GET accessor, or a SET accessor, you cannot force the property to NOT have a particular accessor in an implementing class. However, you can force the accessor to be missing when an instance of the class is used through an interface reference. You can do this by omitting the accessor in the property interface definition. Even though the accessor might be implemented in the class property, it will appear, when used through an interface reference, that the accessor implementation does not exist.

See also

Class-based object reference, CLASS statement, DEFINE DATASET statement, DEFINE EVENT statement, DEFINE PROPERTY statement, DEFINE TEMP-TABLE statement, METHOD statement, Type-name syntax, USING statement

INTERVAL function

Returns the time interval between two DATE, DATETIME, or DATETIME-TZ values as an INT64 value.

Syntax

```
INTERVAL ( datetime1 , datetime2 , interval-unit )
```

datetime1

An expression whose value is a DATE, DATETIME, or DATETIME-TZ.

datetime2

An expression whose value is a DATE, DATETIME, or DATETIME-TZ.

interval-unit

A character constant, or a character expression that evaluates to one of the following time units: 'years', 'months', 'weeks', 'days', 'hours', 'minutes', 'seconds' or 'milliseconds'. These values are case insensitive and might be singular.

Notes

- This function returns a signed integer value (positive or negative). For example, if *datetime1* is less than *datetime2*, the INTERVAL function returns a negative value.
- If *datetime1* or *datetime2* is a DATE or DATETIME, the time value defaults to midnight and the time zone value defaults to the session's time zone, respectively.
- You are responsible for managing value overflow, if any.
- When used with 'months', the INTERVAL function does not take into account that not all months have the same number of days. For example, when determining if an interval between dates in two consecutive months is one full month, the function only checks to see if the day in the date of the later month is greater than or equal to the day in the date of the earlier month. Both of the following function calls return '0', i.e., not a full month, in spite of the fact that the second example involves the calendar end dates of March and April.

```
INTERVAL(04/29/2009,03/30/2009,"months")

INTERVAL(04/30/2009,03/31/2009,"months")
```

See also

- Date subtraction operator, - Datetime subtraction operator, ADD-INTERVAL function, DATE function, DATETIME function, DATETIME-TZ function

IS-ATTR-SPACE function

This function is deprecated.

Syntax

```
IS-ATTR-SPACE
```

See also

TERMINAL statement

IS-CODEPAGE-FIXED function

Returns TRUE if the code page of the specified LONGCHAR variable is fixed; otherwise it returns FALSE.

Syntax

```
IS-CODEPAGE-FIXED ( longchar )
```

longchar

The name of a LONGCHAR variable.

See also

FIX-CODEPAGE statement, GET-CODEPAGE function

IS-COLUMN-CODEPAGE function

Returns TRUE if the specified CLOB field is a COLUMN-CODEPAGE CLOB. Otherwise, it returns FALSE (that is, if the CLOB is a DBCODEPAGE CLOB or a TTCODEPAGE CLOB).

Syntax

```
IS-COLUMN-CODEPAGE ( field )
```

field

The name of a CLOB field.

IS-DB-MULTI-TENANT function

Returns TRUE if a specified database is multi-tenant enabled, and returns FALSE if it is not.

Syntax

```
IS-DB-MULTI-TENANT( [ database-name ] )
```

database-name

A character expression that evaluates to a logical database name or database alias. If no database is specified and more than one database is connected, the AVM raises an error. If the database name is not a valid name for a connected database, the AVM raises an error.

See also

IS-MULTI-TENANT attribute

IS-LEAD-BYTE function

Returns TRUE if the first character of the string is the lead-byte of a multi-byte character. Returns FALSE if it is not.

Syntax

```
IS-LEAD-BYTE ( string )
```

string

A character expression (a constant, field name, variable name, or any combination of these) whose value is a character.

Example

In this example, IS-LEAD-BYTE returns TRUE because the first byte of the first character is the lead-byte of a double-byte character. The output is "Lead: yes":

```
DEFINE VARIABLE Lead AS LOGICAL.
Lead = IS-LEAD-BYTE ("中 xy").
DISPLAY Lead WITH 1 COLUMN.
```

See also

OVERLAY statement, SUBSTRING statement

ISO-DATE function

Returns a character representation of a DATE, DATETIME , or DATETIME-TZ that conforms to the ISO 8601 standard for date/time representations.

Note: These formats are equivalent to the XML Schema `date` and `dateTime` formats.

Syntax

```
ISO-DATE ( expression  )
```

expression

An expression that evaluates to a DATE, DATETIME or DATETIME-TZ.

The ISO-DATE function returns the character string in the standard ISO format of the data type. The following table lists the standard ISO formats for each data type.

Table 44: Standard ISO formats

Data type	ISO format
DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DDTHH:MM:SS.SSS
DATETIME-TZ	YYYY-MM-DDTHH:MM:SS.SSS+HH:MM

See also

ADD-INTERVAL function, DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, ETIME function, INTERVAL function, MONTH function, MTIME function, NOW function, TIME function, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

KBLABEL function

Returns the keyboard label (such as **F1**) of the key that performs a specified ABL function (such as GO).

Note: Does not apply to SpeedScript programming.

Syntax

```
KBLABEL ( key-function )
```

key-function

An expression whose value is the name of the special ABL key function. See *OpenEdge Programming Interfaces* for possible values of *key-name*. If *key-function* is a constant, enclose it in quotation marks (""). See the same guide for a list of key functions and the corresponding standard keyboard keys.

Example

The r-kblabl.p procedure allows the user to update some of the fields in each of the customer records, and displays a message in the status message area at the bottom of the window:

r-kblabl.p

```
STATUS INPUT "Enter data, then press " + KBLABEL("GO").
FOR EACH Customer:
  UPDATE Customer.CustNum Customer.Name Customer.Address Customer.City
    Customer.State.
END.
```

Note

If you reassign a new function key for the key function with the ON statement, the KBLABEL function returns the new key.

KEYCODE function

Evaluates a key label (such as **F1**) for a key in the predefined set of keyboard keys and returns the corresponding = key code (such as 301) as an INTEGER value. See *OpenEdge Programming Interfaces* for a list of key codes and key labels.

Note: Does not apply to SpeedScript programming.

Syntax

```
KEYCODE ( key-label )
```

key-label

A constant, field name, variable name, or expression that evaluates to a character string that contains a key label. If *key-label* is a constant, enclose it in quotation marks (" ").

Example

This procedure displays a menu and highlights different selections on the menu depending on which key you press. On the first iteration of the REPEAT block, the COLOR statement tells the AVM to color msg[ix] with the same color used to display messages. Because the initial value of ix is 1, msg[ix] is the first menu selection. Therefore, the first menu selection is colored MESSAGES.

r-keycod.p

```
DEFINE VARIABLE msg  AS CHARACTER NO-UNDO EXTENT 3.
DEFINE VARIABLE ix   AS INTEGER   NO-UNDO INITIAL 1.
DEFINE VARIABLE newi AS INTEGER   NO-UNDO INITIAL 1.

DISPLAY
  "     Please choose     " SKIP(1)
  " 1  Run order entry    " @ msg[1]
    SKIP
  " 2  Run receivables    " @ msg[2]
    SKIP
  " 3  Exit               " @ msg[3]
    SKIP
  WITH CENTERED FRAME menu NO-LABELS.

REPEAT:
  COLOR DISPLAY MESSAGES msg[ix] WITH FRAME menu.
  READKEY.
  IF LASTKEY = KEYCODE("CURSOR-DOWN") AND ix < 3 THEN
    newi = ix + 1.
  ELSE IF LASTKEY = KEYCODE("CURSOR-UP") AND ix > 1 THEN
    newi = ix - 1.
  ELSE IF LASTKEY = KEYCODE("GO") OR LASTKEY = KEYCODE("RETURN") THEN LEAVE.

  IF ix <> newi THEN
    COLOR DISPLAY NORMAL msg[ix] WITH FRAME menu.
  ix = newi.
END.
```

When you press the cursor-down key, the following occurs:

1. The READKEY statement reads the value of the key you pressed.
2. The first IF . . . THEN . . . ELSE statement tests to see if the key code of the key you pressed is CURSOR-DOWN. It also checks whether the value of `ix` is less than 3. Both of these things are true, so the procedure adds one to the value of `newi`, making `newi` equal two.
3. The next two IF statements are ignored because the condition in the first IF statement was true. The procedure continues on the last IF statement: IF `ix` <> `newi` THEN COLOR DISPLAY NORMAL msg[ix] WITH FRAME menu.
4. Remember, `ix` is still 1 but `newi` is now 2. Thus, `ix` is not equal to `newi`. Which means that the IF statement test is true. Therefore, the AVM colors `msg[ix]`, which is still `msg[1]` (the first menu selection), NORMAL. So the first menu selection is no longer highlighted.
5. Just before the end of the REPEAT block, `ix` is set equal to `newi`. Which means that `msg[ix]]` is now `msg[2]`, or the second menu selection.
6. On the next iteration, the COLOR statement colors `msg[ix]`, that is the second menu selection, MESSAGES. The end result of pressing **CURSOR-DOWN** is that the highlight bar moves to the second menu selection.

See also

KEYFUNCTION function, KEYLABEL function

KEYFUNCTION function

Evaluates an integer expression (such as 301) and returns a character string that is the function of the key associated with that integer expression (such as GO).

Note: Does not apply to SpeedScript programming.

Syntax

```
KEYFUNCTION ( expression )
```

expression

A constant, field name, variable name, or expression whose value is an integer key code.

Example

This procedure displays a menu and highlights different selections, depending on which key you press. On the first iteration of the REPEAT block, the COLOR statement tells the AVM to color msg[ix] with the same color used to display messages. Because the initial value of ix is 1, msg[ix] is the first menu selection. Therefore, the first menu selection is colored MESSAGES.

r-keyfn.p

```
DEFINE VARIABLE msg  AS CHARACTER NO-UNDO EXTENT 3.
DEFINE VARIABLE ix   AS INTEGER   NO-UNDO INITIAL 1.
DEFINE VARIABLE newi AS INTEGER   NO-UNDO INITIAL 1.
DEFINE VARIABLE func AS CHARACTER NO-UNDO.

DISPLAY
  "     Please choose     " SKIP(1)
  " 1  Run order entry    " @ msg[1] SKIP
  " 2  Run receivables    " @ msg[2] SKIP
  " 3  Exit               " @ msg[3] SKIP
  WITH CENTERED FRAME menu NO-LABELS.

REPEAT:
  COLOR DISPLAY MESSAGES msg[ix] WITH FRAME menu.
  READKEY.
  func = KEYFUNCTION(LASTKEY).
  IF func = "CURSOR-DOWN" AND ix < 3 THEN
    newi = ix + 1.
  ELSE IF func = "CURSOR-UP" AND ix > 1 THEN
    newi = ix - 1.
  ELSE IF func = "GO" OR func = "RETURN" THEN LEAVE.
  IF ix <> newi THEN
    COLOR DISPLAY NORMAL msg[ix] WITH FRAME menu.
  ix = newi.
END.
```

See the example in the KEYCODE function reference entry for details on what happens if you press keylabel component.

Notes

- The value returned by the KEYFUNCTION function is affected by any ON statements you use to redefine the value of the key represented by *expression*.
- If the key represented by *expression* has no function currently assigned to it or if it has the function of BELL, KEYFUNCTION returns a null value.
- KEYFUNCTION(-2) is equal to ENDKEY.

See also

KEYCODE function, KEYLABEL function

KEYLABEL function

Evaluates a key code (such as 301) and returns a character string that is the predefined keyboard label for that key (such as **F1**).

Note: Does not apply to SpeedScript programming.

Syntax

```
KEYLABEL ( key-code )
```

key-code

The key code of the key whose label you want to know. A special case of *key-code* is LASTKEY. See *OpenEdge Programming Interfaces* for a list of key codes and key labels.

Example

This procedure reads each keystroke the user makes, leaving the procedure only when the user presses **GO**. The KEYLABEL function tests the LASTKEY pressed, and returns the label of the key. (Remember that the value in LASTKEY is the key code of the last key pressed.)

r-keylbl.p

```
DISPLAY "Press the " + KBLABEL("GO") + " key to leave procedure"
  FORMAT "x(50)".
REPEAT:
  READKEY.
  HIDE MESSAGE.
  IF LASTKEY = KEYCODE(KBLABEL("GO")) THEN RETURN.
  MESSAGE "Sorry, you pressed the"  KEYLABEL(LASTKEY) "key.".
END.
```

Note

Some key codes can be associated with more than one key label. The KEYLABEL function always returns the label listed first in the ABL table of key labels.

See also

KEYCODE function, KEYFUNCTION function

KEYWORD function

Returns a character value that indicates whether a string is an ABL reserved keyword.

Syntax

```
KEYWORD ( expression )
```

expression

A constant, field name, variable name, or expression that results in a character string. If expression matches an ABL reserved keyword or valid abbreviation of a reserved keyword, the KEYWORD function returns the full keyword. If there is no match, the KEYWORD function returns the Unknown value (?).

In some cases, the abbreviation for a keyword is also a keyword. For example, if *expression* is "def" (the abbreviation for DEFINE) or "col" (the abbreviation for COLUMN), the KEYWORD function returns the values "def" and "col", respectively.

If you use the Run-time client, the KEYWORD function always returns the Unknown value (?).

Example

In this example, the KEYWORD function tests the value of formname. If the user tries to use a reserved word as a form name, the AVM displays a message to try again.

r-keywd.p

```
DEFINE VARIABLE formname AS CHARACTER NO-UNDO FORMAT "x(20)".

REPEAT ON ERROR UNDO, RETRY:
  UPDATE formname.
  IF KEYWORD(formname) NE ? THEN DO:
    MESSAGE formname + " may not be used as a form name".
    UNDO, RETRY.
  END.
  ELSE LEAVE.
END.
```

Notes

- Because KEYWORD recognizes abbreviations, it does not distinguish between FORM and FORMAT or between ACCUM and ACCUMULATE.
- This function returns the Unknown value (?) for colors and most data types, as well as all unreserved keywords. See the Keyword Index for a list of ABL reserved and unreserved keywords.
- KEYWORD is less restrictive than the KEYWORD-ALL function. Use this function if you do not want to use ABL reserved keywords as field names, for example.
- For SpeedScript, all ABL reserved keywords are also reserved for SpeedScript.

See also

KEYWORD-ALL function

KEYWORD-ALL function

Returns a character value that indicates whether a string is an ABL keyword. This function returns all keywords and does not distinguish between reserved or unreserved keywords.

Syntax

```
KEYWORD-ALL ( expression )
```

expression

A constant, field name, variable name, or expression that results in a character string. If expression matches an ABL keyword, whether reserved or unreserved or valid abbreviation of a keyword, the KEYWORD-ALL function returns the full keyword. If there is no match, the KEYWORD-ALL function returns the Unknown value (?).

In some cases, the abbreviation for a keyword is also a keyword. For example, if *expression* is "def" (the abbreviation for DEFINE) or "col" (the abbreviation for COLUMN), the KEYWORD function returns the values "def" and "col", respectively.

If you use the Run-time client, the KEYWORD-ALL function always returns the Unknown value (?).

Example

In this example, the KEYWORD-ALL function tests the value of formname. If the user tries to use a keyword as a form name, the AVM displays a message to try again.

r-keywda.p

```
DEFINE VARIABLE formname AS CHARACTER NO-UNDO FORMAT "x(20)".

REPEAT ON ERROR UNDO, RETRY:
  UPDATE formname.
  IF KEYWORD-ALL(formname) NE ? THEN DO:
    MESSAGE formname + "cannot be used as a form name".
    UNDO, RETRY.
  END.
  ELSE LEAVE.
END.
```

Notes

- Because KEYWORD-ALL recognizes abbreviations, it does not distinguish between FORM and FORMAT or between ACCUM and ACCUMULATE.
- This function returns the Unknown value (?) for colors and most data types, as well as all unreserved keywords. See the Keyword Index section for a list of ABL reserved and unreserved keywords.
- For SpeedScript, all ABL reserved keywords are also reserved for SpeedScript.

See also

KEYWORD function

LAST function

Returns a TRUE value if the current iteration of a DO, FOR EACH, or REPEAT . . . BREAK block is the last iteration of that block.

Syntax

```
LAST ( break-group )
```

break-group

The name of a field or expression you named in the block header with the BREAK BY option.

Example

The first FOR EACH block produces a list of the on hand values of the items in inventory. It also automatically generates a total of these on hand values.

The second FOR EACH block does exactly the same thing, except it does not generate the total. Instead, the procedure uses the ACCUMULATE statement and the LAST function. Thus, you can substitute your own labels and formats for the grand total.

r-last.p

```
FOR EACH Item NO-LOCK BY Item.OnHand * Item.Price DESCENDING:
  DISPLAY Item.ItemNum Item.OnHand * Item.Price (TOTAL) LABEL "Value-oh"
    WITH USE-TEXT.
END.

FOR EACH Item NO-LOCK BREAK BY Item.OnHand * Item.Price DESCENDING:
  FORM Item.ItemNum value-oh AS DECIMAL LABEL "Value-oh"
    WITH COLUMN 40 USE-TEXT.
  DISPLAY Item.ItemNum Item.OnHand * Item.Price @ value-oh.
  ACCUMULATE Item.OnHand * Item.Price (TOTAL).
  IF LAST(Item.OnHand * Item.Price) THEN DO:
    UNDERLINE value-oh.
    DISPLAY ACCUM TOTAL Item.OnHand * Item.Price @ value-oh.
  END.
END.
```

See also

FIRST function, FIRST-OF function, LAST-OF function

LASTKEY function

Returns, as an INTEGER value, the integer key code of the most recent event read from the user (that is, from the keyboard or mouse) during an interaction with a procedure.

Note: Does not apply to SpeedScript programming.

Syntax

```
LASTKEY
```

Example

In this procedure, the user can move through the Customer file and update certain fields in each of the Customer records. The GO-ON option tells the procedure to continue on to the following statements if the user presses **F9**, **F10**, or **F12**. To determine what action to take, the LASTKEY function compares the key code of the last key pressed with the key codes **F9**, **F10**, and **F12**.

r-lastky.p

```
DISPLAY "You may update each customer. After making your changes," SKIP
  "Press one of:" SKIP(1)
  KBLABEL("GO") "Make the changes permanent" SKIP
  KBLABEL("END-ERROR")  "Undo changes and exit" SKIP
  "F9" SPACE(7) "Undo changes and try again" SKIP
  "F10" SPACE(6) "Find next customer" SKIP
  "F12" SPACE(6) "Find previous customer"
  WITH CENTERED FRAME instr.

FIND FIRST Customer.

REPEAT:
  UPDATE Customer.CustNum Customer.Name Customer.Address Customer.City
    Customer.State GO-ON(F9 F10 F12) WITH 1 DOWN.
       IF LASTKEY = KEYCODE("F9")  THEN UNDO, RETRY.
  ELSE IF LASTKEY = KEYCODE("F10") THEN FIND NEXT Customer.
  ELSE IF LASTKEY = KEYCODE("F12") THEN FIND PREV Customer.
END.
```

Notes

- The LASTKEY function is double-byte enabled. The LASTKEY function returns values only after the input method places the data in the keyboard buffer. It returns the key code of the most recent key sequence returned from the keyboard buffer. A key sequence is the set of keystrokes necessary to generate one character or function key event in ABL.
- If you used a READKEY statement that timed out (you specified a number of seconds by using the PAUSE option with the READKEY statement), or if a PAUSE statement times out, the value of LASTKEY is -1.
- If you use the PAUSE option with the READKEY statement, the value of LASTKEY is the key you press to end the PAUSE.
- When the ABL session starts, the value of LASTKEY is -1. This value remains the same until the first input, READKEY, or procedure pause occurs. The LASTKEY function is reset to -1 each time you return to the Procedure Editor.
- If you read data from a file, LASTKEY is set to the last character read from the file. For an INSERT, PROMPT-FOR, SET or UPDATE statement, this is always KEYCODE("RETURN"). For a READKEY statement, this is the character read from the file. If you reach past the end of the file, LASTKEY is -2.
- For more information on keys, see *OpenEdge Programming Interfaces*.

See also

READKEY statement

LAST-OF function

Returns a TRUE value if the current iteration of a DO, FOR EACH, or REPEAT . . . BREAK block is the last iteration for a particular value of a break group.

Syntax

```
LAST-OF ( break-group )
```

break-group

The name of a field or expression you named in the block header with the BREAK BY option.

Example

This procedure uses LAST-OF to display a single line of information on each `Item.CatPage` group in the `Item` file, without displaying any individual item data. It produces a report that shows the aggregate value `OnHand` for each catalog page.

r-lastof.p

```
FOR EACH Item NO-LOCK BREAK BY Item.CatPage:
  ACCUMULATE Item.OnHand * Item.Price (TOTAL BY Item.CatPage).
  IF LAST-OF(Item.CatPage) THEN
    DISPLAY Item.CatPage (ACCUM TOTAL BY Item.CatPage
      Item.OnHand * Item.Price) LABEL "Value-oh".
END.
```

See also

FIRST function, FIRST-OF function, LAST function

LC function

Converts any uppercase characters in a CHARACTER or LONGCHAR expression to lowercase characters, and returns the result. The data type of the returned value matches the data type of the expression passed to the function.

Syntax

```
LC ( expression )
```

expression

A constant, field name, variable name, or expression that results in a CHARACTER or LONGCHAR value.

Example

This procedure finds a Customer record. After the user updates the SalesRep field, the procedure converts the first character of the SalesRep value to uppercase and the remaining characters to lowercase.

r-lc.p

```
REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum.
  DISPLAY Customer.Name.
  UPDATE Customer.SalesRep.
  Customer.SalesRep = CAPS(SUBSTRING(Customer.SalesRep, 1, 1) ) +
    LC(SUBSTRING(Customer.SalesRep, 2) ).
  DISPLAY Customer.SalesRep.
END.
```

The CAPS function uses the SUBSTRING function to extract the first character of the field, which it then converts to uppercase.

In the LC function, the result of the SUBSTRING function is the remaining characters in the `SalesRep` field, starting with character position 2. (No length is specified, so the remainder of the string is assumed). The LC function converts these characters to lowercase.

Notes

- The LC function returns lowercase characters relative to the settings of the Internal Code Page (-cpinternal) and Case Table (`-cpcase`) startup parameters. For more information on these parameters, see *Startup Command and Parameter Reference*.
- The LC function is double-byte enabled. The specified expression can yield a string containing double-byte characters; however, the LC function changes only single-byte characters in the string.

See also

CAPS function

LDBNAME function

Returns the logical name of a database that is currently connected.

Syntax

```
LDBNAME
  (
    { integer-expression
    | logical-name
    | alias
    | BUFFER bufname
    }
  )
```

integer-expression

The sequence number of a database the ABL session is connected to. For example, LDBNAME(1) returns information on the first database the ABL session is connected to, LDBNAME(2) returns information on the second database the ABL session is connected to, etc. If you specify a sequence number that does not correspond to a database the ABL session is connected to, the LDBNAME function returns the Unknown value (?).

logical-name or *alias*

These forms of the LDBNAME function require a quoted character string or a character expression as a parameter. If the parameter is the logical name of a connected database or an alias of a connected database then the logical name is returned. Otherwise, the AVM returns the Unknown value (?).

BUFFER *bufname*

The name of a database table or buffer. The BUFFER option lets you determine the database a certain table belongs to without hard-coding the logical database name or alias.

Example

This procedure disconnects all currently connected databases. After a database is disconnected, the connected databases are renumbered to reflect the change. For example, if databases 1, 2, 3, and 4, are connected and the procedure disconnects database 3, database 4 becomes database 3.

r-ldbnm.p

```
DO WHILE LDBNAME(1) <> ? : /* the parameter. is the number 1 */
  DISCONNECT VALUE(LDBNAME(1)).
END.
```

Note

To determine if a particular name is an ALIAS or a logical database name, use the following procedure:

r-tstnm.p

```
DEFINE VARIABLE testnm AS CHARACTER NO-UNDO.

SET testnm.
IF LDBNAME(testnm) = testnm THEN
  MESSAGE testnm "is a true logical database name.".
ELSE IF LDBNAME(testnm) = ? THEN
  MESSAGE testnm "is not the name or alias of any connected database.".
ELSE
  MESSAGE testnm "is an ALIAS for database " LDBNAME(testnm).
```

See also

CONNECT statement, CONNECTED function, CREATE ALIAS statement, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, NUM-DBS function, PDBNAME function

LE or < = operator

Returns a TRUE value if the first of two expressions is less than or equal to the second.

Syntax

```
expression { LE | <= } expression
```

expression

A constant, field name, variable name, or expression. The expressions on either side of the LE or < = must be of the same data type, although one can be integer and the other decimal.

Example

This procedure lists all the items with zero or negative on-hand quantities:

r-le.p

```
FOR EACH Item WHERE Item.OnHand <= 0:
  DISPLAY Item.ItemNnum Item.ItemName Item.OnHand.
END.
```

Notes

- By default, the AVM uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (`-cpcoll`) startup parameter take precedence over a collation specified for any database the AVM accesses during the session, except when the AVM uses or modifies pre-existing indexes. If you do not specify a collation with the `-cpcoll` startup parameter, the AVM uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, the AVM uses the collation rules with the default name "`basic`" (which might or might not exist in the `convmap.cp` file).
- If either of the expressions is the Unknown value (?), then the result is the Unknown value (?); if both of the expressions are the Unknown value (?), then the result is TRUE.
- You can compare character strings with LE. Most character comparisons are case insensitive in ABL. That is, upper-case and lower-case characters have the same sort value. However, it is possible to define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If either *`expression`* is a field or variable defined as case sensitive, the comparison is case sensitive and "Smith" does not equal "smith".
- Characters are converted to their sort code values for comparison. Using the default case-sensitive collation table, all uppercase letters sort before all lowercase letters (for example, a is greater than Z, but less than b.) Note also that in character code uppercase A is less than [, \ , ^ , _, and ' , but lowercase a is greater than these.
- You cannot compare data of different DATE, DATETIME, and DATETIME-TZ data types to each other using LE. You must first convert different date and datetime data types to the same data type before doing a comparison between them.

- You can use LE to compare a LONGCHAR variable to another LONGCHAR or CHARACTER variable. The variable values are converted to `-cpinternal` for comparison and must convert without error, or the AVM raises a run-time error.
- You cannot use LE to compare any of the LOB data types, including BLOB or CLOB.
- You can use LE to compare two enums, including both ABL and .NET enums, as long as they are the same enum type. LE compares the underlying numerical values of the enumerators.

LEAVE statement

Exits from a block. Execution continues with the first statement after the end of the block.

Syntax

```
LEAVE [ label ]
```

label

The name of the block you want to leave. If you do not name a block, the AVM leaves the innermost iterating block that contains the LEAVE statement. If there is no such block, then the AVM leaves the procedure block.

Example

This procedure represents part of a menu program. If the user chooses N, P, F, or Q, the procedure leaves the inner choose block and goes on to process the menu selection. If the user presses any other key, the procedure rings the terminal bell.

r-leave.p

```
DEFINE VARIABLE valid-choice AS CHARACTER NO-UNDO INITIAL "NPFQ".
DEFINE VARIABLE selection    AS CHARACTER NO-UNDO FORMAT "x".

main-loop:
REPEAT:
  choose:
  REPEAT ON ENDKEY UNDO choose, RETURN:
    MESSAGE "(N)ext (P)rev (F)ind (Q)uit"
      UPDATE selection AUTO-RETURN.
    /* Selection was valid */
    IF INDEX(valid-choice, selection) <> 0 THEN LEAVE choose.
    BELL.
  END.  /* choose */

  /* Processing for menu choices N, P, F here */
  IF selection = "Q" THEN LEAVE main-loop.
END. /* REPEAT */
```

See also

NEXT statement, RETURN statement, UNDO statement

LEFT-TRIM function

Removes leading white space, or other specified characters, from a CHARACTER or LONGCHAR expression. The data type of the returned value matches the data type of the expression passed to the function.

Syntax

```
LEFT-TRIM ( expression [ , trim-chars ] )
```

expression

An expression (a constant, field name, variable name, or expression) whose value is a CHARACTER or LONGCHAR. If *expression* is a case-sensitive variable, the AVM performs a case-sensitive trim. If *expression* is a LONGCHAR, the result is in the same code page.

trim-chars

A character expression that specifies the characters to be trimmed from *expression*. If you do not specify *trim-chars*, the LEFT-TRIM function removes spaces, tabs, line feeds, and carriage returns.

Example

The following example shows the effect of the TRIM, LEFT-TRIM, and RIGHT-TRIM functions on a string value:

r-ltrim.p

```
DEFINE VARIABLE ix  AS INTEGER   NO-UNDO.
DEFINE VARIABLE txt AS CHARACTER NO-UNDO FORMAT "X(26)"
  INITIAL "***** This is a test *****".

DEFINE BUTTON b_left  LABEL "Left Trim".
DEFINE BUTTON b_right LABEL "Right Trim".
DEFINE BUTTON b_trim  LABEL "Trim".
DEFINE BUTTON b_quit  LABEL "Quit" AUTO-ENDKEY.

DEFINE FRAME butt-frame
  txt ix LABEL "String Length" SKIP(2)
  b_left b_right b_trim b_quit
WITH CENTERED TITLE "Original Text String".

DEFINE FRAME trimed-frame
  txt LABEL "Trimed Text"
  ix  LABEL "Length"
WITH CENTERED.

ON CHOOSE OF b_trim, b_right, b_left IN FRAME butt-frame DO:
  FRAME trimed-frame:TITLE = "Data After " + SELF:LABEL.
  DISPLAY TRIM(txt, "* ") WHEN SELF:LABEL = "Trim" @ txt
    LENGTH(TRIM(txt, "* ")) WHEN SELF:LABEL = "Trim" @ ix
    LEFT-TRIM(txt,"* ") WHEN SELF:LABEL = "Left Trim" @ txt
    LENGTH(LEFT-TRIM(txt,"* ")) WHEN SELF:LABEL = "Left Trim" @ ix
    RIGHT-TRIM(txt, "* ") WHEN SELF:LABEL = "Right Trim" @ txt
    LENGTH(RIGHT-TRIM(txt, "* ")) WHEN SELF:LABEL = "Right Trim" @ ix
  WITH FRAME trimed-frame.
END.

ENABLE b_left b_right b_trim b_quit WITH FRAME butt-frame.

ix = LENGTH(txt).
DISPLAY txt ix WITH FRAME butt-frame.

WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame.
```

Notes

- The LEFT-TRIM function is similar to the TRIM function except that it trims characters only from the left end of the string.
- If *expression* is a case-sensitive field or variable, then *trim-chars* is also as case sensitive. Otherwise, *trim-chars* is not case sensitive.
- The LEFT-TRIM function is double-byte enabled. The specified *expression* and *trim-chars* arguments can contain double-byte characters. LEFT-TRIM does not remove double-byte space characters by default.

See also

RIGHT-TRIM function, TRIM function

LENGTH function

Returns, as an INTEGER value, the number of characters, bytes, or columns in a string, an expression of type RAW, or a BLOB field.

Syntax

```
LENGTH ( { exp | raw-exp | blob-field } [ , type ] )
```

exp

An expression that evaluates to a CHARACTER, LONGCHAR, INTEGER, LOGICAL, or CLOB type. If *exp* evaluates to an INTEGER or LOGICAL, the STRING function is used to first convert it to a character string before returning the LENGTH. For a LOGICAL, YES/TRUE returns 3 and NO/FALSE returns 2.

raw-exp

A function or variable name that returns a RAW value.

blob-field

An expression that evaluates to a BLOB field.

type

A character expression that indicates whether you want the length of *exp* in character units, bytes, or columns. A double-byte character registers as one character unit. By default, the unit of measurement is character units.

There are three valid types: "CHARACTER," "RAW," and "COLUMN." The expression "CHARACTER" indicates that the length is measured in characters, including double-byte characters. The expression "RAW" indicates that the length is measured in bytes. The expression "COLUMN" indicates that the length is measured in display or print character-columns. If you specify the *type* as a constant expression, ABL validates the type specification at compile time. If you specify the *type* as a non-constant expression, the AVM validates the type specification at run time.

Note: The expression "COLUMN" is not valid for a LONGCHAR variable or a CLOB field.

Examples

This procedure produces a report that contains item information. Because the information on the report fills the entire width of the screen, this procedure shortens the information in the description field for each item. If the description of an item is longer than eight characters, the procedure converts the description to the first eight characters followed by ellipses.

r-length.p

```
DEFINE VARIABLE short-name AS CHARACTER NO-UNDO FORMAT "x(11)" LABEL "Desc".

FOR EACH Item NO-LOCK:
  IF LENGTH(Item.ItemName, "COLUMN") > 8 THEN
    short-name = SUBSTRING(Item.ItemName,1,8, "COLUMN") + "..." .
  ELSE
    short-name = Item.ItemName.
  DISPLAY Item.ItemNum short-name Item.OnHand Item.Allocated
    Item.ReOrder Item.OnOrder Item.Price FORMAT "$>>>9.99".
END.
```

In this procedure, the LENGTH function returns the number of bytes in the Name of number 29. The procedure returns a 15, the number of bytes in the Name, Bug in a Rug-by.

r-rawlen.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

FIND Customer NO-LOCK WHERE Customer.CustNum = 29.
ix = LENGTH(Customer.Name, "RAW").
DISPLAY Customer.Name ix LABEL "Byte Length".
```

Note

If the value of the expression is the Unknown value (?), the LENGTH function returns the Unknown value (?).

LENGTH statement

Changes the number of bytes in a raw variable.

Syntax

```
LENGTH ( variable ) = expression
```

variable

A variable of type RAW.

expression

An expression that returns an integer.

Note: You can use compound assignment operators (+=, -=, *=, /=) with the LENGTH statement. For more information, see += Addition assignment operator, -= Subtraction assignment operator, *= Multiplication assignment operator, and /= Division assignment operator.

Example

This procedure takes the number of bytes in the name stored in the variable r1 and truncates it to 2 bytes:

r-rawln1.p

```
/* You must connect to a non-OpenEdge demo database to run this procedure */
DEFINE VARIABLE r1 as RAW NO-UNDO.

FIND Customer NO-LOCK WHERE Customer.CustNum = 29.
r1 = RAW(Customer.Name).
LENGTH(r1) = 2.
```

The following example code uses the *= assignment operator to double the number of bytes for the variable.

```
/* You must connect to a non-OpenEdge demo database to run this procedure */
VAR RAW r1.

FIND Customer NO-LOCK WHERE Customer.CustNum = 29.
r1 = RAW(Customer.Name).
LENGTH(r1) *= 2.
```

Notes

- If *variable* is the Unknown value (?), it remains the Unknown value (?).
- If *expression* is greater than the number of bytes in *variable*, the AVM appends null bytes so that the length of *variable* equals the length of *expression*.

LIBRARY function

Parses a character string in the form *path-name*<<*member-name*>>, where *path-name* is the pathname of an ABL r-code library and *member-name* is the name of a file within the library, and returns the pathname of the library. The double angle brackets indicate that *member-name* is a file in a library. If the string is not in this form, the LIBRARY function returns the Unknown value (?).

Typically, you use the LIBRARY function with the SEARCH function to retrieve the name of a library. The SEARCH function returns character strings of the form *path-name*<<*member-name*>> if it finds a file in a library.

Syntax

```
LIBRARY ( string )
```

string

A character expression whose value is the pathname of a file in a library. In Windows, the pahtname cannot contain characters outside of the non-Unicode code page. See *Internationalize ABL Applications* for more information about Unicode and code pages.

Example

This procedure searches for a file that you specify. It displays a message indicating whether the file is not found in your path, is found in a library within your path, or is found in your path but not in a library.

r-rlib.p

```
DEFINE VARIABLE what-lib AS CHARACTER NO-UNDO.
DEFINE VARIABLE location AS CHARACTER NO-UNDO.
DEFINE VARIABLE myfile   AS CHARACTER NO-UNDO FORMAT "x(16)"
  LABEL "R-code File".

SET myfile.
location = SEARCH(myfile).

IF location = ?THEN DO:
  MESSAGE "Can't find" myfile.
  LEAVE.
END.

what-lib = LIBRARY(location).
IF what-lib <> ? THEN
  MESSAGE myfile "can be found in library" what-lib.
ELSE
  MESSAGE myfile "is not in a library but is in" location.
```

Note

You can improve the performance of an application by using the SEARCH and LIBRARY functions for any files you want to execute that you specify with a relative pathname. For example, if you want to execute a procedure file several times that you specify with a relative pathname, you can return the absolute pathname for the file using the SEARCH and LIBRARY functions, then execute the RUN statement with that absolute pathname, which avoids any need for the RUN statement to search PROPATH each time.

See also

MEMBER function, SEARCH function

LINE-COUNTER function

Returns the current line number of paged output as an INTEGER value.

The initial value of LINE-COUNTER is 1. At the completion of each DISPLAY statement, the AVM increments LINE-COUNTER by the number of lines that were output in that DISPLAY statement. LINE-COUNTER continues to increase until after at least one line has been printed on a new page.

LINE-COUNTER returns a 0 if the output is not paged.

Syntax

```
LINE-COUNTER [ ( stream | STREAM-HANDLE handle ) ]
```

stream

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream. For more information on streams, see this book's DEFINE STREAM statement reference entry and *OpenEdge Programming Interfaces*.

STREAM-HANDLE *handle*

Specifies the handle to a stream. If *handle* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

Example

This procedure prints a Customer report, categorized by State. At the end of each State category, it tests to see if there are at least four lines left on the page. The LINE-COUNTER function returns the current line number of output. If that number plus four is greater than the total number of lines on the page (returned by the PAGE-SIZE function), then the procedure starts the new page. If there are four or more lines left, the procedure skips a line before printing the next customer record.

r-linec.p

```
OUTPUT TO PRINTER.
FOR EACH Customer BREAK BY Customer.State:
  DISPLAY Customer.CustNum Customer.Name Customer.Address Customer.City
    Customer.State.
  IF LAST-OF(Customer.State) THEN DO:
    IF LINE-COUNTER + 4 > PAGE-SIZE THEN PAGE.
    ELSE DOWN 1.
  END.
END.
```

Notes

- When output is sent to a device other than the terminal screen, the AVM defers displaying a frame until another frame is displayed. That way, if you display the same frame several times consecutively, the AVM performs all those displays at once. Because of this optimization, if the last display fills the page, the value returned by the LINE-COUNTER function can be larger than the page size, even though the next frame is displayed at the start of the new page.
- Use a procedure like this one to verify that output is positioned on the first non-header line of a new page:

```
DEFINE VARIABLE newpage AS LOGICAL NO-UNDO INITIAL TRUE.

DEFINE STREAM output1.

FOR EACH Customer NO-LOCK:
  FORM HEADER "Page Header" PAGE-NUMBER(output1) "Line"
    LINE-COUNTER(output1)
    WITH FRAME one PAGE-TOP NO-LABELS NO-BOX.

  VIEW STREAM output1 FRAME one.
  DISPLAY STREAM output1 name PAGE-NUMBER(output1) LINE-NUMBER(output1)
    WITH NO-LABELS NO-BOX.

  IF new-page THEN
    DISPLAY STREAM output1 "First Line".
  newpage = IF LINE-COUNTER(output1) > PAGE-SIZE(output1) THEN
    TRUE ELSE FALSE.
END.
```

See also

DEFINE STREAM statement, Stream object handle

LIST-EVENTS function

Returns a comma-separated list of the valid events for a specified object or widget.

Note: Does not apply to SpeedScript programming.

Syntax

```
LIST-EVENTS ( handle [ , platform ] )
```

handle

A handle to a valid object or widget. The function returns a list of the events that are valid for that object or widget.

platform

A character-string value that specifies a display type. Valid values are GUI and TTY. Some events are valid only on certain platforms. If you omit the *platform* parameter, the AVM uses the platform for the current session.

Example

The following example uses the LIST-EVENTS function to populate a selection list with all the valid events for a widget. When you run this procedure, type ? at any time to see a list of valid events for the widget that currently has focus.

r-levent.p

```
DEFINE VARIABLE inv-price   NO-UNDO LIKE item.price.
DEFINE VARIABLE inv-value   NO-UNDO LIKE item.price.
DEFINE VARIABLE report-type AS INTEGER   NO-UNDO INITIAL 1.

DEFINE VARIABLE event-list  AS CHARACTER NO-UNDO
  VIEW-AS SELECTION-LIST INNER-CHARS 20 INNER-LINES 5 SCROLLBAR-VERTICAL.

DEFINE BUTTON ok-butt     LABEL "OK" AUTO-GO.
DEFINE BUTTON cancel-butt LABEL "CANCEL" AUTO-ENDKEY.

FORM
  inv-price LABEL "Price" AT ROW 1.25 COLUMN 2
  report-type LABEL "Report Sorted ..." AT ROW 2.25 COLUMN 2
    VIEW-AS RADIO-SET RADIO-BUTTONS
    "By Catalog Page", 1, "By Inventory Value", 2 SKIP
  ok-butt cancel-butt
  WITH FRAME select-frame SIDE-LABELS.

FORM event-list
  WITH FRAME list-frame NO-LABELS TITLE "Events" WIDTH 30.

ON ? ANYWHERE DO:
  FRAME list-frame:TITLE = "Events for " + FOCUS:TYPE.
  event-list:LIST-ITEMS IN FRAME list-frame = LIST-EVENTS(FOCUS).
  DISPLAY event-list WITH FRAME list-frame.
  ENABLE event-list WITH FRAME list-frame.
  RETURN NO-APPLY.
END.

ENABLE ALL WITH FRAME select-frame.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

See also

LAST-EVENT system handle, LIST-QUERY-ATTRS function, LIST-SET-ATTRS function, LIST-WIDGETS function, VALID-EVENT function

---

LIST-QUERY-ATTRS function

Returns a comma-separated list of attributes and methods that are supported for an object or widget.

Syntax

```
LIST-QUERY-ATTRS ( handle )
```

handle

A handle to a valid object or widget. The function returns a list of the attributes and methods that are supported for that object or widget.

Example

The following example uses the LIST-QUERY-ATTRS and LIST-SET-ATTRS functions to populate selection lists with the valid attributes and methods for a specified widget. When you run this procedure, type ? at any time to see lists of valid attributes for the widget that currently has focus.

r-lattrs.p

```
DEFINE VARIABLE inv-price   NO-UNDO LIKE item.price.
DEFINE VARIABLE inv-value   NO-UNDO LIKE item.price.
DEFINE VARIABLE report-type AS INTEGER   NO-UNDO INITIAL 1.
DEFINE VARIABLE qattr-list  AS CHARACTER NO-UNDO LABEL "Readable"
  VIEW-AS SELECTION-LIST INNER-CHARS 20 INNER-LINES 5
  SCROLLBAR-VERTICAL SORT.
DEFINE VARIABLE wattr-list  AS CHARACTER NO-UNDO LABEL "Writable"
  VIEW-AS SELECTION-LIST INNER-CHARS 20 INNER-LINES 5
  SCROLLBAR-VERTICAL SORT.

DEFINE BUTTON ok-butt LABEL "OK" AUTO-GO.
DEFINE BUTTON cancel-butt LABEL "CANCEL" AUTO-ENDKEY.

FORM
  inv-price LABEL "Price" AT ROW 1.25 COLUMN 2
  report-type LABEL "Report Sorted ..." AT ROW 2.25 COLUMN 2
    VIEW-AS RADIO-SET RADIO-BUTTONS
    "By Catalog Page", 1, "By Inventory Value", 2 SKIP
  ok-butt cancel-butt
  WITH FRAME select-frame SIDE-LABELS.

FORM qattr-list wattr-list
  WITH FRAME list-frame TITLE "Attributes" WIDTH 30 COLUMN 47.

ON ? ANYWHERE DO:
  FRAME list-frame:TITLE = "Attributes for " + FOCUS:TYPE.
  qattr-list:LIST-ITEMS IN FRAME list-frame = LIST-QUERY-ATTRS(FOCUS).
  wattr-list:LIST-ITEMS IN FRAME list-frame = LIST-SET-ATTRS(FOCUS).
  ENABLE qattr-list wattr-list WITH FRAME list-frame.
  RETURN NO-APPLY.
END.

ENABLE ALL WITH FRAME select-frame.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

See also

CAN-QUERY function, CAN-SET function, LAST-EVENT system handle, LIST-EVENTS function, LIST-SET-ATTRS function, LIST-WIDGETS function, VALID-EVENT function

LIST-SET-ATTRS function

Returns a comma-separated list of attributes that can be set for an object or widget.

Syntax

```
LIST-SET-ATTRS ( handle )
```

handle

A handle to a valid object or widget. The function returns a list of the attributes that can be set for that object or widget.

Example

For an example of the LIST-SET-ATTRS function, see the LIST-QUERY-ATTRS function reference entry.

Note

When used with the SECURITY-POLICY handle, this function does not return the XCODE-SESSION-KEY writeable attribute because this attribute contains an encryption key for encrypted source code.

See also

CAN-QUERY function, CAN-SET function, LAST-EVENT system handle, LIST-EVENTS function, LIST-QUERY-ATTRS function, LIST-WIDGETS function, VALID-EVENT function

LIST-WIDGETS function

Returns a comma-separated list of objects and widget types that respond to a specified event.

Note: Does not apply to SpeedScript programming.

Syntax

```
LIST-WIDGETS ( event-name [ , platform ] )
```

event-name

A character-string expression that evaluates to an event name.

platform

A character-string value that specifies a display type. Valid values are GUI and TTY. Some events are valid only on certain platforms. If you omit the *platform* parameter, the AVM uses the platform for the current session.

Example

The following example prompts for an event name and then displays a list of widget types that support that event:

r-lwids.p

```
DEFINE VARIABLE event-name  AS CHARACTER NO-UNDO FORMAT "x(24)"
  LABEL "Event".
DEFINE VARIABLE widget-list AS CHARACTER NO-UNDO LABEL "Widgets"
  VIEW-AS SELECTION-LIST INNER-CHARS 24 INNER-LINES 6 SCROLLBAR-VERTICAL.

FORM event-name SKIP widget-list
  WITH FRAME main-frame SIDE-LABELS.

REPEAT WITH FRAME main-frame:
  DISABLE widget-list.
  SET event-name.
  widget-list:LIST-ITEMS = LIST-WIDGETS(event-name).
  DISPLAY widget-list.
  ENABLE widget-list.
  PAUSE.
END.
```

See also

LAST-EVENT system handle, LIST-EVENTS function, LIST-QUERY-ATTRS function, LIST-SET-ATTRS function, VALID-EVENT function

LOAD statement
(Windows only)

Creates application defaults, involving colors, fonts, environment variables, etc., or loads existing defaults, to a graphical or character application.

Specifically, the LOAD statement:

- Creates registry keys and initialization file entries
- Creates new initialization files
- Loads entries from the registry or from an existing initialization file

For more information on application defaults, see the topics on colors and fonts in *OpenEdge Programming Interfaces*.

Note: Does not apply to SpeedScript programming.

Syntax

```
LOAD environment
   [ DIR directory ]
   [ APPLICATION ]
   [ NEW ]
   [ BASE-KEY { key-name | "INI" } ]
   [ NO-ERROR ]
```

environment

A CHARACTER expression that evaluates to one of the following:

- The name of a registry key to create
- The name of an initialization file to create
- The name of an existing registry key
- The name of an existing initialization file

DIR *directory*

A CHARACTER expression that evaluates to the directory path of one of the following:

- An existing initialization file
- An initialization file to create

If you omit this option, LOAD looks for an existing initialization file, or creates a new initialization file, in the working directory. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

APPLICATION

Has no effect; supported only for backward compatibility.

NEW

Creates a new registry key or a new initialization file. If the key or file already exists, LOAD overwrites its data.

BASE-KEY *key-name* | "INI"

Bypasses the standard search rules.

If you specify BASE-KEY *key-name*, LOAD looks for or creates the environment in the registry only under base key *key-name*.

If you specify BASE-KEY "INI" (the quotes are required), LOAD looks for or creates only the initialization file *environment*.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

See the USE statement reference entry for an example.

Notes

- If you specify LOAD *environment*, LOAD searches for a registry key and for an existing initialization file, and tries to load one or the other. The search logic, which assumes that *environment* has the format *path**rootname*.*extension* (where *path* and *extension* are optional) and that *version* is the current OpenEdge version, is as follows:
  - Search the registry under HKEY_CURRENT_USER for *path**rootname*.*extension*.

    If found, load it.

- Else search the registry under HKEY_CURRENT_USER for *SOFTWARE\PSC\PROGRESS\version\path\rootname.extension.*

  If found, load it.

- Else search the registry under HKEY_CURRENT_USER for *SOFTWARE\path\rootname.extension*.

  If found, load it.

- Else search the registry under HKEY_CURRENT_USER for *rootname.*

  If found, load it.

- Else search the registry under HKEY_CURRENT_USER for *SOFTWARE\PSC\PROGRESS\version\rootname*.

  If found, load it.

- Else search the registry under HKEY_CURRENT_USER for *SOFTWARE\rootname.*

  If found, load it.

- Else search the registry under HKEY_LOCAL_MACHINE for *path\rootname.extension.*

  If found, load it.

- Else search the registry under HKEY_LOCAL_MACHINE for *SOFTWARE\PSC\PROGRESS\version\path\rootname.extension*.

  If found, load it.

- Else search the registry under HKEY_LOCAL_MACHINE for *SOFTWARE\path\rootname.extension.*

  If found, load it.

- Else search the registry under HKEY_LOCAL_MACHINE for *rootname.*

  If found, load it.

- Else search the registry under HKEY_LOCAL_MACHINE for *SOFTWARE\PSC\PROGRESS\version\rootname.*

  If found, load it.

- Else search the registry under HKEY_LOCAL_MACHINE for *SOFTWARE\rootname.*

  If found, load it.

- Else search for the initialization file *path\rootname.extension*.

  If found, load it.

- Else, error.

- If you specify LOAD *environment* BASE-KEY *key-name*, where *key-name* is the name of a registry base key, LOAD loads the registry key *key-name\environment.*

  Registry base keys are as follows:

  - HKEY_CLASSES_ROOT
  - HKEY_CURRENT_CONFIG (Win95 and NT 4.0)
  - HKEY_CURRENT_USER
  - HKEY_DYN_DATA (Win95 and NT 4.0)

- HKEY_LOCAL_MACHINE
- HKEY_USERS

- If you specify LOAD *environment* BASE-KEY "INI", LOAD loads the initialization file *environment*.
- If you specify LOAD *environment* NEW, LOAD creates a new key in the registry under HKEY_CURRENT_USER and names the new key *environment*.
- If you specify LOAD *environment* NEW BASE-KEY *key-name*, LOAD creates a new key in the registry under *key-name* and names the new key *environment*.
- If you specify LOAD *environment* NEW BASE-KEY "INI", LOAD creates a new initialization file and names it *environment.ini*.
- To change the application environment, load defaults using the LOAD statement, make them current using the USE statement, then access them using the GET-KEY-VALUE and PUT-KEY-VALUE statements.

See also

GET-KEY-VALUE statement, LOAD statement, NO-ERROR option, PUT-KEY-VALUE statement, UNLOAD statement, USE statement

---

LOAD-PICTURE statement
(Windows only; Graphical interfaces only)

Returns a COM-HANDLE to an OlePictureObject. You can use this COM-HANDLE to set graphical properties of controls.

Note: Does not apply to SpeedScript programming.

Syntax

```
LOAD-PICTURE [ image ]
```

image

A CHARACTER expression representing the name of the graphical file. This file can have one of the following extensions: .BMP, .EMF, .ICO, .CUR, .DIB. If the filename is not fully qualified, LOAD-PICTURE searches for a matching file on the user's path. The filename can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

Example

The following program fragment illustrates the use of the LOAD-PICTURE statement:

```
DEFINE VARIABLE chPic     AS COM-HANDLE NO-UNDO.
DEFINE VARIABLE chControl AS COM-HANDLE NO-UNDO.

/* Get the COM-HANDLE of an Image control */

/* Use LOAD-PICTURE to get a COM-HANDLE to the bitmap contained in myPic.bmp */
chPic = LOAD-PICTURE("myPic.bmp").

/* Set the control's Picture property for the bitmap */
chControl:Picture = chPic.
```

LOCKED function

Returns a TRUE value in the following cases:

- a record is not available to a prior FIND . . . NO-WAIT statement because another user has locked the record
- a record is not available to a GET . . . NO-WAIT statement because another user has locked the record

Syntax

```
LOCKED record
```

record

The name of a record or buffer.

To use the LOCKED function with a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

Example

The FIND statement in this procedure tries to retrieve a customer record according to a supplied customer number. Because of the NO-ERROR option, the FIND statement does not return an error if it cannot find the record. The NO-WAIT option causes FIND to return immediately if the record is in use by another user.

r-locked.p

```
REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum NO-ERROR NO-WAIT.
  IF NOT AVAILABLE Customer THEN DO:
    IF LOCKED Customer THEN
      MESSAGE "Customer record is locked".
    ELSE
      MESSAGE "Customer record was not found".
    NEXT.
  END.
  DISPLAY Customer.CustNum Customer.Name Customer.City Customer.State.
END.
```

A record might not be available if it is locked (being used by another user) or does not exist. The LOCKED function returns a TRUE value if the record is locked. In this case, the `r-locked.p` procedure displays a message that the record is locked. If the record is not locked, the procedure displays a message that the record does not exist.

Note: The result of the LOCKED function depends on the lock mode specified. For example, if the FIND statement uses SHARE-LOCK and no user has an EXCLUSIVE-LOCK on the record, the LOCKED function returns FALSE. If the FIND statement uses SHARE-LOCK and another user has an EXCLUSIVE-LOCK on the record, the LOCKED function returns TRUE. The current copy of the record in the buffer to which the LOCKED function applies is not locked. Rather, LOCKED refers to an error condition that can occur if the record is locked. Consider using the NO-WAIT and NO-ERROR options on the FIND statement to return immediately and raise an error condition.

See also

AMBIGUOUS function, AVAILABLE function, FIND statement, NEW function (record buffers)

LOG function

Calculates the logarithm of an expression using a specified base and returns that logarithm as a DECIMAL value.

Syntax

```
LOG ( expression [ , base ] )
```

expression

A decimal expression that you want the logarithm of.

base

A numeric expression that is the base you want to use. If you do not specify a base, LOG returns the natural logarithm, base (e). The *base* must be greater than 1.

Example

This procedure prompts the user for a base and a number, and then displays the log of the number. The VALIDATE option on the UPDATE statement ensures that the user enters a base value greater than 1 and a number greater than 0.

r-log.p

```
DEFINE VARIABLE base   AS DECIMAL NO-UNDO FORMAT ">>>,>>>.9999".
DEFINE VARIABLE number AS DECIMAL NO-UNDO.

REPEAT:
  UPDATE base VALIDATE(base > 1, "Base must be greater than 1").
  REPEAT:
    UPDATE number VALIDATE(number > 0, "Number must be positive").
    DISPLAY number LOG(number, base) LABEL "LOG(NUMBER, BASE)".
  END.
END.
```

Notes

- The LOG function is accurate to approximately 10 decimal places.
- After converting the base and exponent to floating-point format, the LOG function uses standard system routines. On some machines, the logarithm routines do not handle large numbers well and might cause your terminal to hang.

LOGICAL function

Converts any data type into the LOGICAL data type.

Syntax

```
LOGICAL ( expression [ , char-expression-format ] )
```

expression

An *expression* in the data type that you want to convert to logical.

char-expression-format

A character expression that evaluates to a valid logical format, such as "si/no", or "da/nyet". This argument is ignored unless *expression* is of CHARACTER type. Also, this argument is only needed if *expression* evaluates to something other than the usual TRUE or FALSE, or YES or NO values.

Example

The following code fragment illustrates the Logical function:

```
DEFINE VARIABLE mychar AS CHARACTER NO-UNDO.
DEFINE VARIABLE v-log  AS LOGICAL   NO-UNDO.

mychar = "si".

v-log = LOGICAL(mychar, "si/no")
/* v-log is TRUE */
```

Notes

- If the value of *expression* is the Unknown value (?), the LOGICAL function returns the Unknown value (?).
- If *expression* is of type DECIMAL, INTEGER, INT64, DATE, DATETIME, DATTIME-TZ, or HANDLE, the function returns TRUE if the value of *expression* is nonzero. If the value of *expression* is 0, it returns FALSE. The second argument is ignored if present.
- If *expression* is of type LONGCHAR or CHARACTER, it returns TRUE or FALSE depending on the value in the expression and the format used. Whether or not *char-expression-format* is given, the case-insensitive values TRUE, FALSE, YES, NO, abbreviated to 1 character, are always accepted. For example, a "Y" is interpreted as TRUE.
- If *char-expression-format* is given, it is validated. If it is not valid, an error message appears and the Unknown value (?) is returned. Otherwise, the format is used to interpret the character string if it is not one of the following: TRUE, FALSE, YES, or NO. For example, LOGICAL ("si", "si/no") returns TRUE.
- Data types such as RAW, MEMPTR, LVARBINARY, and so on return the Unknown value (?), but this is not considered an error.

See also

STRING function, INTEGER function, DATE function

Logical values

Represent values of logical expressions.

Syntax

```
{ [ YES | TRUE ] | [ NO | FALSE ] }
```

YES | TRUE

A value that signifies a valid result for a logical expression.

NO | FALSE

A value that signifies an invalid result for a logical expression.

Note

You must use these values in a procedure even if alternate values are given in the FORMAT specification for a field or variable.

LOOKUP function

Returns an INTEGER value giving the position of an expression in a list. Returns a 0 if the expression is not in the list.

Syntax

```
LOOKUP ( expression , list [ , character ] )
```

expression

A constant, field name, variable name, or expression that results in a character value that you want to look up within a list of character expressions. If the value of *expression* is the Unknown value (?), the result of the LOOKUP function is the Unknown value (?).

list

A character expression that contains the expression you name with the *expression* argument. Each entry in the list is separated with a delimiter. The list can be a variable of type CHARACTER or LONGCHAR. If *list* contains the Unknown value (?), LOOKUP returns the Unknown value (?).

character

A delimiter you define for the list. The default is a comma. This allows the LOOKUP function to operate on non-comma-separated lists.

Examples

This procedure prompts the user for a New England state. The LOOKUP function tests the value against the list of states stored in the stlist variable. If there is no match (the result is 0), the procedure displays a message. Otherwise, the procedure prompts the user for another New England state.

r-lookup.p

```
DEFINE VARIABLE stlist AS CHARACTER NO-UNDO
  INITIAL "ME,MA,VT,RI,CT,NH".
DEFINE VARIABLE state  AS CHARACTER NO-UNDO FORMAT "x(2)".

REPEAT:
  SET state LABEL "Enter a New England state, 2 characters".
  IF LOOKUP(state, stlist) = 0 THEN
    MESSAGE "This is not a New England state".
END.
```

The following example uses a different delimiter, which list all fields that have "sls" or "sales" as words in their standard Dictionary labels:

r-look2.p

```
FOR EACH _Field
  WHERE LOOKUP("sls",_Field._Label," ") > 0
     OR LOOKUP("sales",_Field._Label," ") > 0:

  DISPLAY _Field._Field-Name _Field._Label.
END.
```

Notes

- If *expression* contains a delimiter, LOOKUP returns the beginning of a series of entries in *list*. For example, LOOKUP("a,b,c","x,a,b,c") returns a 2.
- Most character comparisons are case insensitive in ABL. By default, upper-case and lower-case characters have the same sort value. However, you can define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If the *expression* or *list* is defined as case sensitive, the comparison between them is also case sensitive and "Smith" does not equal "smith".
- The LOOKUP function is double-byte enabled. The specified *expression* can yield a string value that contains double-byte characters and the *character* delimiter can be a double-byte character.
- The LOOKUP function always returns 0 if an expression is equal to a delimiter.

See also

ENTRY function, ENTRY statement, INDEX function, NUM-ENTRIES function

LT or < operator

Returns a TRUE value if the first of two expressions is less than the second.

Syntax

```
expression { LT | < } expression
```

expression

A constant, field or variable name, or expression. The expressions on either side of the LT or < = must be the same data type, although one can be an integer and the other decimal.

Example

This procedure displays information for those `Item` records whose `OnHand` value is less than the `Allocated` value:

r-lt.p

```
FOR EACH Item NO-LOCK WHERE Item.OnHand < Item.Allocated:
  DISPLAY Item.ItemNum Item.ItemName Item.OnHand Item.Allocated.
END.
```

Notes

- By default, the AVM uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (`-cpcoll`) startup parameter take precedence over a collation specified for any database the AVM accesses during the session, except when the AVM uses or modifies pre-existing indexes. If you do not specify a collation with the `-cpcoll` startup parameter, the AVM uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, the AVM uses the collation rules with the default name "`basic`" (which might or might not exist in the `convmap.cp` file).
- If either of the expressions is the Unknown value (?), then the result is the Unknown value (?); if both of the expressions are the Unknown value (?), then the result is FALSE.
- You can compare character strings with LT. Most such comparisons are case insensitive in ABL. That is, upper-case and lower-case characters have the same sort value. It is possible to define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If either *`expression`* is a field or variable defined as case sensitive, the comparison is case sensitive and "Smith" does not equal "smith."
- Characters are converted to their sort code values for comparison. Using the default case-sensitive collation table, all uppercase letters sort before all lowercase letters (for example, a is greater than Z, but less than b.) Note also that in character code uppercase A is less than [, \ , ^ , _, and ' , but lowercase a is greater than these.
- You cannot compare data of different DATE, DATETIME, and DATETIME-TZ data types to each other using LT. You must first convert different date and datetime data types to the same data type before doing a comparison between them.
- You can use LT to compare a LONGCHAR variable to another LONGCHAR or CHARACTER variable. The variable values are converted to `-cpinternal` for comparison and must convert without error, or the AVM raises a run-time error.
- You cannot use LT to compare one CLOB field to another.
- You can use LT to compare two enums, including both ABL and .NET enums, as long as they are the same enum type. LT compares the underlying numerical values of the enumerators.

MATCHES operator

Compares a character expression to a pattern and evaluates to a TRUE value if the expression satisfies the pattern criteria.

Syntax

```
expression MATCHES pattern
```

`expression`

A CHARACTER or LONGCHAR expression that you want to check to see if it conforms with the *`pattern`*.

`pattern`

A character expression that you want to match with the string. This can include a constant, field name, variable name, or expression whose value is a character.

The *pattern* can contain wildcard characters: a period (.) in a particular position indicates that any single character is acceptable in that position; an asterisk (*) indicates that any group of characters is acceptable, including a null group of characters.

Example

This procedure displays customer information for all Customers whose Address ends in St. The procedure does not use an index for the Customer search in r-match.p.

r-match.p

```
FOR EACH Customer NO-LOCK WHERE Customer.Address MATCHES("*St"):
  DISPLAY Customer.Name Customer.Address Customer.City Customer.State
    Customer.Country.
END.
```

Notes

- MATCHES does not use index information when performing a comparison; it always scans the entire data table.
- MATCHES does not ignore trailing blanks as does the equal (EQ) comparison operator. Thus, "abc" does not match "abc " although they are considered equal.
- Most character comparisons are case insensitive in ABL. By default, all characters are converted to uppercase prior to comparisons. However, you can define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If the *expression* preceding the MATCHES keyword is a field or variable defined as case sensitive, the comparison is case sensitive. In a case-sensitive comparison "SMITH" does not equal "Smith".
- MATCHES converts a LONGCHAR variable value to -cpinternal prior to comparison. The variable must convert without error, or the AVM raises a run-time error.
- You cannot use MATCHES to compare one CLOB field to another.
- To specify either an asterisk (*), as a literal character rather than a wildcard character in the pattern, or a period (.) as a literal character, you must enter a tilde (~) before the character. However, if you specify the match pattern as a literal quoted string in a procedure file, you must enter each tilde as a double tilde (~ ~). The first tilde escapes the
- second tilde, so that the AVM interprets the second tilde as a tilde for the match pattern. For example, the result of "*a.b" MATCHES "~~*a~~.b" is TRUE.
- The MATCHES function is double-byte enabled. Both the specified *expression* and *pattern* arguments can contain double-byte characters.

See also

BEGINS operator

MAXIMUM function

Compares two or more values and returns the largest value.

Syntax

```
MAXIMUM ( expression , expression [ , expression ] ... )
```

expression

A constant, field name, variable name, or expression. If there is a mixture of decimal and integer data types, decimal type is returned.

Example

In this procedure, if the `CreditLimit` value is under 20,000, the procedure adds 10,000 to that value. Otherwise, the procedure sets `CreditLimit` to 30,000. The MAXIMUM function determines the greater of the original `CreditLimit` value and the new cred-lim2 value.

r-maxmum.p

```
DEFINE VARIABLE cred-lim2 AS DECIMAL NO-UNDO FORMAT ">>,>>9.99".

FOR EACH Customer NO-LOCK:
  cred-lim2 = IF Customer.CreditLimit < 20000 THEN
    Customer.CreditLimit + 10000 ELSE 30000.
  DISPLAY Customer.CreditLimit cred-lim2
    MAXIMUM(cred-lim2, Customer.CreditLimit)
      LABEL "Maximum of these two values".
END.
```

Notes

- When comparing character values, if at least one of the character fields is defined as case sensitive, then MAXIMUM treats all of the values as case sensitive for the sake of the comparisons. If none of the values is case sensitive, MAXIMUM treats lowercase letters as if they were uppercase letters.
- You cannot compare data of different DATE, DATETIME, and DATETIME-TZ data types to each other using MAXIMUM. You must first convert different date and datetime data types to the same data type before doing a comparison between them.

See also

MINIMUM function

MD5-DIGEST function

Hashes the specified data using the RSA Message Digest Hash Algorithm (MD5), and returns a 16-byte binary message digest value as a RAW value.

Syntax

```
MD5-DIGEST( data-to-hash [ , hash-key ] )
```

data-to-hash

The source data to hash. The data may be of type CHARACTER, LONGCHAR, RAW, or MEMPTR. If the data is a CHARACTER or LONGCHAR value, the AVM converts it to UTF-8 (which ensures a consistent value regardless of code page settings). To avoid this automatic conversion, specify a RAW or MEMPTR value.

hash-key

An optional key value to use in the hash operation. The key may be of type CHARACTER, LONGCHAR, RAW, or MEMPTR. If the key is a CHARACTER or LONGCHAR value, the AVM converts it to UTF-8 (which ensures a consistent value regardless of code page settings). To avoid this automatic conversion, specify a RAW or MEMPTR value. This key value is combined with the source data before the hash operation begins.

If the *hash-key* value contains a null character, the null character is included in the hash operation.

Note: This function can be used for backward compatibility but should not be used in new development.

See also

MESSAGE-DIGEST function, SHA1-DIGEST function

MEMBER function

Parses a reference to a member of an ABL r-code library and returns the simple member name.

Syntax

```
MEMBER ( string )
```

string

A character expression (a constant, field name, variable or expression that results in a character value) whose value is the pathname of a file in an r-code library.

The MEMBER function parses a character string in the form *path-name*<<*member-name*>>, where *path-name* is the pathname of a library and *member-name* is the name of a file within the library, and returns *member-name*. The double angle brackets indicate that *member-name* is a file in a library. If the string is not in this form, the MEMBER function returns the Unknown value (?).

Use the MEMBER function with the SEARCH function to determine whether a file is in a library. If a data file is in a library, you must first extract the file from the library in order to read it. (See *Manage ABL Applications* for more information on extracting a file from a library.) The SEARCH function returns a character string in the form *path-name*<<*member-name*>> if it finds a file in a library.

Example

This procedure prompts for the name of a file. Using this value, the procedure searches for the file. If it does not find the file, it displays a message and ceases operation. If it does find the file, it tests to see if the file is in a library. If so, the procedure displays the filename and the name of the library. Otherwise, the procedure displays the pathname of the file returned by SEARCH.

r-memb.p

```
DEFINE VARIABLE what-lib AS CHARACTER NO-UNDO.
DEFINE VARIABLE location AS CHARACTER NO-UNDO.
DEFINE VARIABLE myfile   AS CHARACTER NO-UNDO FORMAT "x(16)"
  LABEL "R-code File".

SET myfile.
location = SEARCH(myfile).

IF location = ? THEN DO:
  MESSAGE "Can't find" myfile.
  LEAVE.
END.

what-lib = LIBRARY(location).
IF what-lib <> ? THEN
  MESSAGE MEMBER(location) "can be found in library" what-lib.
ELSE
  MESSAGE myfile "is not in a library but is in" location.
```

See also

LIBRARY function, SEARCH function

MESSAGE statement

Displays messages in the message area at the bottom of the window or in an alert box (or in an output stream—see the Notes section). By default, an area at the bottom line of the window is reserved for ABL system messages. An area above that is reserved for messages you display with the MESSAGE statement.

Syntax

```
MESSAGE
  [ COLOR color-phrase ]
  { expression | SKIP [ ( n ) ] } ...
  [ VIEW-AS ALERT-BOX
      [alert-type]
      [ BUTTONS button-set ]
      [ TITLE title-string ]
  ]
  [ { SET | UPDATE } field
          { AS datatype | LIKE field }
          [ FORMAT string ]
          [ AUTO-RETURN ]
  ]
  [ IN WINDOW window ]
```

COLOR *color-phrase*

Displays a message using the color you specify with the COLOR phrase.

```
  NORMAL
| INPUT
| MESSAGES
| protermcap-attribute
| dos-hex-attribute
| { [ BLINK- ] [ BRIGHT- ] [ fgnd-color ] [ bgnd-color ] }
| { [ BLINK- ] [ RVV- ] [ UNDERLINE- ] [ BRIGHT- ]
      [ fgnd-color ] }
| VALUE ( expression )
```

For more information on *color-phrase*, see the COLOR phrase reference entry.

Note: The COLOR phrase does not have any effect in a Windows environment.

expression

An expression (a constant, field name, variable name, or expression) whose value you want to display in the message area. If *expression* is not character, it is converted to character before it is displayed. If you do not use this option, you must use either the SET or UPDATE option.

SKIP [(*n*)]

Indicates a number (*n*) of blank lines to insert into the message. The value of *n* can be 0. If you do not specify *n*, or if *n* is 0, a new line is started unless the current position is already the start of a new line.

You can only use this option with the VIEW-AS ALERT-BOX option.

VIEW-AS ALERT-BOX [*alert-type*]

Specifies that the message is displayed in an alert box rather than in the window message area. The value of *alert-type* determines the type of alert box. The possible values are:

- MESSAGE
- QUESTION
- INFORMATION
- ERROR
- WARNING

The type of alert box affects the visual representation of the box.

BUTTONS *button-set*

Specifies what sets of buttons are available within the alert box. The possible button sets are as follows:

- YES-NO
- YES-NO-CANCEL
- OK
- OK-CANCEL
- RETRY-CANCEL

The name of each button set indicates the buttons in that set. For example, YES-NO contains two buttons labeled YES and NO; YES-NO-CANCEL contains three buttons labeled YES, NO, and CANCEL; OK contains a single button labeled OK. If you do not specify a button set, the default is OK.

`TITLE` *`title-string`*

Specifies a value to display in the title bar of the alert box.

`SET` *`field`*

Displays the *`expression`* you specified and SETs the field or variable you name. (It prompts the user for input and assigns the value entered to the field or variable.) You cannot test the field with the ENTERED function or the NOT ENTERED function.

`UPDATE` *`field`*

Displays the *`expression`* you specified and updates the field or variable you name. (It displays the current value of the field or variable, prompts for input, and assigns the value entered in the field or variable.) You cannot test the field with the ENTERED function or the NOT ENTERED function. For an alert box, *`field`* must be a LOGICAL variable. It sets the default button and returns the user's choice. If the alert box has two buttons, they represent the values TRUE and FALSE, respectively. If the alert box has three buttons, they represent the values TRUE, FALSE, and the Unknown value (?), respectively.

`AS` *`datatype`*

Defines *`field`* as a variable of type *`datatype`*. You must use this option or the LIKE option if *`field`* has not been previously defined.

`LIKE` *`field`*

Defines the field specified in SET or UPDATE as a database field or a previously defined variable.

`FORMAT` *`string`*

The format that you want to use to display the *`field`* used in the SET or UPDATE option.

For more information on display formats, see *ABL Essentials*.

If you do not use the FORMAT option, ABL uses the defaults shown in the following table.

Table 45: Default display formats

Type of expression	Default format
Field	Format from schema
Variable	Format from variable definition

Type of expression	Default format
Constant character	Length of character string
Other	Default format for the data type of the expression

The following table shows the default formats for the Other expression.

Table 46: Default data type display formats

Data type	Default display format
CHARACTER	x(8)
CLASS[40]	N/A
DATE	99/99/99
DATETIME	99/99/9999 HH:MM:SS.SSS
DATETIME-TZ	99/99/9999 HH:MM:SS.SSS+HH:MM
DECIMAL	->>,>>9.99
HANDLE[41]	>>>>>>9
INT64	->,>>>,>>9
INTEGER	->,>>>,>>9
LOGICAL	yes/no
MEMPTR[42]	See footnote [42].
RAW[42]	See footnote [42].
RECID	>>>>>>9
ROWID[42]	See footnote [42].

`AUTO-RETURN`

Performs a carriage return when the field that is SET or UPDATEd is full.

[40] For a class instance, ABL automatically invokes the ToString( ) method (Object) (provided by the Progress.Lang.Object class) on the object reference. Instead, you can also first explicitly convert the object reference to a displayable type using the INT64 function, the INTEGER function, or the STRING function.

[41] To display a HANDLE, you must first convert it using either the INT64 or INTEGER function and display the result.

[42] You cannot display a MEMPTR, RAW, or ROWID value directly. However, you can convert it to a character string representation using the STRING function and display the result. A ROWID value converts to a hexadecimal string, "0x*`hexdigits`*", where *`hexdigits`* is any number of characters "0" through "9" and "A" through "F". A MEMPTR or RAW value converts to decimal integer string.

IN WINDOW *window*

Specifies the window in which the message is displayed.

Examples

In this procedure, if you enter the number of a Customer that does not exist, the procedure displays a message telling you the Customer does not exist. If the Customer does exist, the procedure displays the Name and SalesRep of the Customer.

r-msg.p

```
REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum NO-ERROR.
  IF NOT AVAILABLE Customer THEN DO:
    MESSAGE "Customer with CustNum " INPUT Customer.CustNum
      " does not exist.  Please try another".
    UNDO, RETRY.
  END.
  ELSE
    DISPLAY Customer.Name Customer.SalesRep.
END.
```

The following example uses two alert boxes:

r-altbox.p

```
DEFINE VARIABLE cust-list AS CHARACTER NO-UNDO
  VIEW-AS SELECTION-LIST SINGLE SIZE 50 BY 10 LABEL "Customers".
DEFINE VARIABLE ok-status AS LOGICAL   NO-UNDO

FORM cust-list
  WITH FRAME sel-frame.

ON DEFAULT-ACTION OF cust-list DO:
  MESSAGE "You have chosen to delete" cust-list:SCREEN-VALUE + "." SKIP(1)
    "Do you really want to delete this customer?"
    VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO-CANCEL
    TITLE "" UPDATE lChoice AS LOGICAL.
  CASE lChoice:
    WHEN TRUE THEN /* Yes */ DO:
      FIND Customer WHERE Customer.Name = cust-list:SCREEN-VALUE
        EXCLUSIVE-LOCK.
      DELETE Customer.
    END.
    WHEN FALSE THEN /* No */ DO:
      MESSAGE "Deletion canceled."
        VIEW-AS ALERT-BOX INFORMATION BUTTONS OK.
      RETURN NO-APPLY.
    END.
    OTHERWISE /* Cancel */ STOP.
  END CASE.
END.

FOR EACH Customer NO-LOCK BY Customer.Name:
  ok-status = cust-list:ADD-LAST(Customer.Name).
END.

ENABLE cust-list WITH FRAME sel-frame.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

In `r-altbox.p`, each time you select an item from the selection list, the procedure displays an alert box to ask if you want to delete the customer. If you choose the No button, then another alert box informs you that the record was not deleted.

Notes

- The MESSAGE statement always sends messages to the current output destination. If the INPUT source is the terminal, the AVM displays messages in the window and also sends them to the current output destination. Compiler error messages also follow this convention.
- If you don't want messages sent to the current output destination, redirect the output to a named stream. The AVM never writes messages to a named stream.

  If you want to send output to a destination other than the terminal, and you do not want messages to appear on the terminal (and if you are not using the terminal as an input source), use one of the statements in the following table.

  Table 47: Suppressing messages to the terminal

Operating system	Input from
UNIX	INPUT FROM /dev/null
Windows	INPUT FROM NUL

  Be sure to use the INPUT CLOSE statement to close the input source.
- The AVM automatically clears messages after any user interaction, such as a SET, UPDATE, or PAUSE statement, but not after a READKEY statement.
- In Microsoft Windows, the message text in VIEW-AS ALERT-BOX is limited to 511 bytes. If the text is longer than 511 bytes, it is truncated.
- When you use the MESSAGE SET or MESSAGE UPDATE statement to update a field, the AVM does not process any validation criteria defined for that field in the database. For example, if the validation criteria for the customer.name field is as follows:

```
Valexp: name BEGINS "a"
```

  Use this statement:

```
MESSAGE UPDATE name
```

  The AVM lets you enter any data, including data that does not start with the letter a, into the name field.

  Use the MESSAGE statement to display a message, but use the SET statement or UPDATE statement to let the user change the data in a frame rather than in the message area.

- If you are displaying a message to the message line and the combination of the text and field you name in a MESSAGE UPDATE statement exceeds the length of the message line, the AVM truncates the text to fit on the message line. For example:

```
DEFINE VARIABLE myvar AS CHARACTER NO-UNDO FORMAT "x(60)".

MESSAGE "abcdefghijklmnopqrstuvwxyz" UPDATE myvar.
```

  Here, the combination of the message text and the myvar variable exceeds 80 characters, so the AVM truncates the message text.

- Using the MESSAGE statement to display decimal values results in truncating the nonsignificant zeros to the right of the decimal point. For example:

```
DEFINE VARIABLE amt AS DECIMAL NO-UNDO FORMAT ">>9.99" INITIAL 1.20.

MESSAGE "Total" amt.
```

  The previous procedure displays the following message:

```
"Total 1.2"
```

  Use functions such as STRING and DECIMAL to control the format of a display.

- If the APPL-ALERT-BOXES attribute of the SESSION system handle is TRUE, then all your messages are displayed in alert boxes. You can also direct all system messages to alert boxes by setting the SYSTEM-ALERT-BOXES attribute of the SESSION system handle to true. You can remove the message area for a window by setting its MESSAGE-AREA attribute to FALSE before it is realized.
- If you use the SET or UPDATE options in a graphical environment, the AVM automatically displays the message as an alert box.
- By default, all text in an alert box is displayed on a single line. If you want to break lines within the text, you must explicitly insert SKIP options into the message.
- If you use the OUTPUT TO statement to divert ABL error and warning messages to an output stream, ABL also diverts messages from the MESSAGE statement the same way. For more information, see the OUTPUT TO statement reference entry.
- The MESSAGE statement does not raise ERROR if the AVM fails to resolve an output expression. Instead, the empty string will be displayed at the specified output location. However, if the block containing the MESSAGE statement contains a CATCH block, then the MESSAGE statement will raise ERROR and you can handle that error with a compatible CATCH block.
- You can use the MESSAGE statement with an object reference for a class instance. In this case, the MESSAGE statement implicitly calls the ToString( ) method of the class to convert the specified object reference to a character value before it displays the result.
- For SpeedScript, the only valid options are: *expression* and SKIP.

See also

COLOR phrase, DECIMAL function, Format phrase, INTEGER function, MESSAGE-LINES function, STRING function

MESSAGE-DIGEST function

Hashes any of several types of source data using the specified hashing algorithm, and returns a RAW message digest value whose size and security depends on the algorithm.

Syntax

```
MESSAGE-DIGEST( hash-algorithm, data-to-hash [, hash-key ] )
```

hash-algorithm

A character string that specifies the hashing algorithm to use to hash the data specified by *data-to-hash*. You can specify one of the following options, which are generally in the order of increased security and decreased performance:

- "MD5" — RSA Message Digest Hash Algorithm, which returns a 16-byte RAW binary message digest value. This is provided for backward compatibility only and should not be used in new development.
- "SHA-1" — United States Government Secure Hash Algorithm, which returns a RAW 20-byte binary message digest value.
- "SHA-256" — United States Government Secure Hash Algorithm, which returns a RAW 32-byte binary message digest value.
- "SHA-512" — United States Government Secure Hash Algorithm, which returns a RAW 64-byte binary message digest value.
- "HMAC-SHA-1" — United States Government HMAC (Hash-based Message Authentication Code) algorithm, which returns a RAW 20-byte binary message digest value.
- "HMAC-SHA-256" — United States Government HMAC (Hash-based Message Authentication Code) algorithm, which returns a RAW 32-byte binary message digest value.
- "HMAC-SHA-384" — United States Government HMAC (Hash-based Message Authentication Code) algorithm, which returns a RAW 48-byte binary message digest value.
- "HMAC-SHA-512" — United States Government HMAC (Hash-based Message Authentication Code) algorithm, which returns a RAW 64-byte binary message digest value.

data-to-hash

The source data to hash. The data may be of type CHARACTER, LONGCHAR, RAW, or MEMPTR. If the data is a CHARACTER or LONGCHAR value, the AVM converts it to UTF-8 (which ensures a consistent value regardless of code page settings). To avoid this automatic conversion, specify a RAW or MEMPTR value.

hash-key

A key value to use in the hash operation for the hashing algorithms. The key may be of type CHARACTER, LONGCHAR, RAW, or MEMPTR. If the key is a CHARACTER or LONGCHAR value, the AVM converts it to UTF-8 (which ensures a consistent value regardless of code page settings). To avoid this automatic conversion, specify a RAW or MEMPTR value. This key value is used along with the source data during the hash operation.

For the HMAC algorithms, it is strongly recommended that you specify a hash key (preferably a randomly generated one).

If *hash-key* is not specified it defaults to the empty string ("").

If either *data-to-hash* or *hash-key* is the Unknown value (?), then the Unknown value (?) is returned from the function.

If the *hash-key* value contains a null character, the null character is included in the hash operation.

Example

The following example demonstrates how to use the MESSAGE-DIGEST function.

```
DEFINE VARIABLE mydata AS CHARACTER NO-UNDO INITIAL "My data to hash".
DEFINE VARIABLE hashkey AS CHARACTER NO-UNDO INITIAL "My hash key".
DEFINE VARIABLE mdvalue AS RAW NO-UNDO.

mdvalue = MESSAGE-DIGEST("SHA-512", mydata, hashkey).

MESSAGE "LENGTH: " LENGTH(mdvalue) SKIP
        "VALUE: " STRING(mdvalue) VIEW-AS ALERT-BOX.
```

See also

MD5-DIGEST function, SHA1-DIGEST function

MESSAGE-LINES function

Returns, as an INTEGER value, the number of lines in the message area at the bottom of the window.

Note: Does not apply to SpeedScript programming.

Syntax

```
MESSAGE-LINES
```

Example

The following example displays a message on each available message line:

r-messl.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

DO ix = 1 TO MESSAGE-LINES:
  MESSAGE "This is message line" ix.
END.
```

METHOD statement

Defines a method of a class, declares a method prototype in an ABL interface, or overrides a method inherited from an ABL or .NET super class. A method is a class member that, when invoked by name, can execute code and return a value similar to a user-defined function. The description that follows begins with the general syntax for defining a method.

Note: This statement is applicable only when used in a class or interface definition (`.cls`) file.

Syntax

```
METHOD [ PRIVATE | PACKAGE-PRIVATE | PROTECTED | PACKAGE-PROTECTED | PUBLIC ]
  [ STATIC | ABSTRACT ] [ OVERRIDE ][ FINAL ]
  { VOID |return-type}method-name
  ( [ parameter [ , parameter ]...] ) :

  method-body
```

Use the following syntax to declare an interface method prototype:

```
METHOD [ PUBLIC ]
  { VOID |return-type}method-name
  ( [ parameter [ , parameter ]...] ) .
```

Use the following syntax to declare an abstract method prototype:

```
METHOD [ PACKAGE-PRIVATE | PROTECTED | PACKAGE-PROTECTED | PUBLIC ]
  [ OVERRIDE ] ABSTRACT
  { VOID |return-type}method-name
  ( [ parameter [ , parameter ]...] ) .
```

[PRIVATE | PACKAGE-PRIVATE | PROTECTED | PACKAGE-PROTECTED | PUBLIC]

Specifies the access mode for this method.

A PRIVATE method can be called by the defining class. An instance can call the private method of another instance if they are both from the same class. A PACKAGE-PRIVATE method can be called from within the class and any class within its package. A PROTECTED method can be called by the defining class and any of its derived classes. A PACKAGE-PROTECTED method can be called from within the class, any class within its package, and from within any subclass that inherits the class. A PUBLIC method can be called from any code (in a class or a procedure) that has access to a class instance.

The default access mode is PUBLIC.

When declaring an interface method prototype, the access mode for this method must be PUBLIC (the default).

When defining an abstract method, the access mode for the method cannot be PRIVATE.

If this method is defined with the OVERRIDE option, the access mode must not be more restrictive than the access mode defined for the overridden ABL or .NET super class method. The following table shows what access mode you can use on an OVERRIDE method in an ABL class when the method being overridden is .NET:

.NET	ABL
protected	PROTECTED, PACKAGE-PROTECTED, PUBLIC
protected internal	PROTECTED, PACKAGE-PROTECTED, PUBLIC
public	PUBLIC

[STATIC]

Defines a method that is a static member of the class type for which it is defined and that is scoped to the ABL session where it is referenced. You can call an accessible static method in any piece of code.

Without this option, ABL defines an instance method that is scoped to a single instance of the class where it is defined. You can call any public instance method (abstract or non-abstract) in any procedure, or in any instance or static method defined inside or outside of the class where the instance property is defined. Any static method can call the public instance method only using an object reference to a class instance that defines the instance method as a member. If the referencing static method is defined in the same class as the public instance method, the class must instantiate itself in order to have access to an instance reference.

For more information on the mechanism for calling methods of different access modes and scopes, see the reference entry for Class-based property access.

The STATIC option is not valid when you:

- Define or implement an abstract method
- Declare an interface method prototype
- Implement an interface method

Note: The access mode modifier and the STATIC/ABSTRACT modifier can appear in either order, e.g., `METHOD STATIC PUBLIC VOID myMethod():...` is valid syntax.

[ABSTRACT]

Defines the method as an abstract member of the class type for which it is defined. The defining class type must also be abstract. If you define an abstract method, it has the following requirements:

- You must specify the OVERRIDE option if an inherited method (abstract or otherwise) has the same name and signature.
- You can specify any access mode other than PRIVATE.
- The abstract method must be overridden and implemented in a derived class.

Any class definition for an instance method that includes the OVERRIDE option and does not include the ABSTRACT option defines a method implementation. If it also includes the ABSTRACT option, the inherited method remains or is redefined as abstract.

This option is not valid either when you define a static method or you declare an interface method prototype.

[`OVERRIDE` **]**

Specifies one of the following:

- This instance method overrides the behavior of another instance method inherited from an ABL or .NET class.
- This instance method implements, or overrides the behavior of, an abstract method inherited from an ABL or .NET abstract class.
- This instance method redefines as abstract an implemented method inherited from an ABL or .NET class.

Note: A .NET abstract class is defined in C# with the `abstract` keyword.

- This static method redefines the behavior of another static method implemented in a super class.

When you specify OVERRIDE, the method signature must match the overridden method with respect to the name, return type, and the number, types, and modes of its parameters. In addition, the access mode cannot be more restrictive than the access mode of the super class method it overrides. When overriding a .NET method, the return type and parameters must also map appropriately to the inherited .NET method return type and parameters. For more information, see the description of the *`return-type`* and *`parameter`* options.

If you are overriding an inherited abstract method and you specify the ABSTRACT option, your overriding method is also defined as abstract, and it must be implemented in a class derived from the defining class. Note that you do not have to override an inherited abstract method that you want to remain abstract as long as the inheriting class is also abstract. However, doing so allows you to specify a less restrictive access mode for the abstract method.

If you do not specify the ABSTRACT option when overriding an inherited abstract method, your method definition implements the abstract method.

If you are overriding an inherited method that already has an implementation and you specify the ABSTRACT option, your overriding method redefines the inherited method as abstract, and it must, again, be implemented in a class derived from the defining class. Note that to redefine an inherited method as abstract that already has an implementation, the defining class must also be abstract.

Note that instance methods do not override static methods and static methods do not redefine instance methods. For more information about overriding and redefining methods, see *Develop Object-oriented ABL Applications*.

You can only override methods of a .NET super class that, in C# terms, are:

- Declared as `virtual` or `abstract`
- Declared as `public`, `protected` or `protected internal`
- Not declared as `static` or `sealed` (FINAL)

This option is not valid when you declare an interface method prototype.

[FINAL]

Indicates this method cannot be overridden or redefined by a method defined in an inheriting subclass.

This option is not valid either when you define an abstract method or when you declare an interface method prototype.

VOID

Indicates this method does not return a value.

return-type

Indicates the data type of the method return value. You can specify *return-type* as one of the following data types. For more information on each data type, see the Data types reference entry:

```
{ CHARACTER | COM-HANDLE | DATE | DATETIME | DATETIME-TZ
  | DECIMAL | HANDLE | INT64 | INTEGER | LOGICAL | LONGCHAR
  | MEMPTR | RAW | RECID | ROWID | AS-data-type
  | [ CLASS ] object-type-name } [ EXTENT [ constant ] ]
```

AS-data-type

If you are defining the return type for a method that overrides a .NET super class method (abstract or otherwise) or implements a method defined in a .NET interface, *return-type* must specify the exact return type of the overridden or implemented .NET method. For a .NET mapped data type that is a default match for an ABL primitive type, you must use the default matching ABL data type, as shown in Table 24: Implicit mappings between .NET and ABL data types. (For example, INTEGER indicates a .NET System.Int32.) For a .NET mapped data type that is **not** a default match for one of the ABL primitive types, ABL provides a data type keyword (the *AS-data-type*) that you must use to explicitly indicate the required .NET data type, as shown in Table 25: Explicit mappings between ABL and .NET data types. (For example, UNSIGNED-BYTE indicates a .NET System.Byte.)

Note: At run time, a method return type defined using an *AS-data-type* keyword behaves in ABL like the corresponding ABL primitive type shown in Table 25: Explicit mappings between ABL and .NET data types. (For example, an UNSIGNED-BYTE behaves like an INTEGER.)

Also note that when overriding or implementing a .NET array return value, you must specify the .NET array object type (for example, `"System.Int32[]"` or `"System.Byte[]"`); you **cannot** use an ABL array equivalent (for example, INTEGER EXTENT or UNSIGNED-BYTE EXTENT).

object-type-name

Specifies the type name of an ABL or .NET class or interface. Specify an object type name using the syntax described in the Type-name syntax reference entry. With an appropriate USING statement, you can also specify a class or interface name alone, without the qualifying package or namespace.

If you are defining the return type for a method that overrides a .NET super class method (abstract or otherwise), or that implements a method defined in a .NET interface, *object-type-name* must specify the exact return type of the overridden or implemented .NET method. However, for .NET inner (nested) type, note the difference in the ABL syntax, which replaces the corresponding period (.) in the .NET object type with a plus (+) (see the Type-name syntax reference entry).

Also note that when overriding or implementing a .NET array return value, you must specify the .NET array object type (for example, `"System.Drawing.Point[]"`); you **cannot** use an ABL array equivalent (such as System.Drawing.Point EXTENT).

[CLASS **]**

If the specified class or interface type name conflicts with an abbreviation of a built-in primitive type name, such as INT for INTEGER, you must specify the CLASS keyword.

For a class or interface return value, ABL returns an object reference associated with the class or interface, not a class instance itself. For more information on object references, see the Class-based object reference reference entry.

[EXTENT **[** *constant* **]]**

Defines the return value as an array of data elements with a primitive or object data type. This option can specify an array return value as either determinate (has a defined number of elements) or indeterminate (has an undefined number of elements). To define a determinate array return value, specify the EXTENT option with the *constant* argument. This optional argument is an integer value that represents the number of elements in the array. To define an indeterminate array return value, specify the EXTENT option without the *constant* argument. For more information about determinate and indeterminate arrays, see Work with one-dimensional arrays in *Basic ABL*.

If you do not use the EXTENT option (or you specify *constant* as 0), the return value is not an array return value.

method-name

The method name. This name must be unique among all methods defined in the class hierarchy of the defining class, unless the method:

- Includes the OVERRIDE option to override an inherited method.
- Overloads a method in the defining class or in any super class within its inherited class hierarchy. In this case, the parameter list must differ from that of every other method with the same name in the defining class, and it must differ from that of every other method with the same name defined in every super class of the defining class.

Note: Members of a class are grouped into six namespaces, including buffers/temp-tables, methods, variables/properties/events, ProDataSets, queries, and data-sources. Methods defined as members of a class share the same namespace. There can be only one class member in this namespace with a given name. If the method is defined in a class that is derived from a .NET class, other restrictions on *method-name* apply. For more information, see the Notes in this reference entry.

(**[** *parameter* **[** , *parameter* **]...]**)

Defines zero or more parameters of the method.

If this method is one of several overloaded methods, the parameter list must be unique among all methods defined in the class hierarchy of the defining class. This uniqueness can be established using a different combination of number, data types, or modes for the parameters. Note that both instance and static methods overload one another within the class hierarchy.

If this method implements an interface method prototype or overrides an inherited method (using the OVERRIDE option), the parameter list must match the parameter list of the interface method or the method this is overriding.

If the method implements a .NET interface method prototype or overrides an inherited .NET method (abstract or otherwise), you must explicitly map any .NET mapped types specified for parameters of the .NET method (see Table 25: Explicit mappings between ABL and .NET data types). Note that at run time, any explicitly mapped parameter behaves like the corresponding ABL primitive type.

For more information on the syntax of *parameter*, establishing uniqueness for method overloading, and on mapping .NET parameter types, see the Parameter definition syntax reference entry.

method-body

The body for a method implementation. For an interface method prototype or an abstract method, you cannot specify *method-body*. For more information about declaring method prototypes in an interface, see the INTERFACE statement reference entry.

Define *method-body* using the following syntax:

```
method-logic
      .
      .
      .
    [catch-block[catch-block...]]
    [finally-block]
END [ METHOD ] .
```

method-logic

The logic of the method. This logic can contain the ABL statements that are allowed within a procedure block, including class-related statements.

If you define the method to return a value of the data type specified by *return-type*, you can execute the RETURN statement to set a value of that data type to return at run time. If *return-type* is defined as a .NET array of mapped types (for example, "System.Byte[]"), you must return an object reference of the specified .NET array of mapped types in the RETURN statement. You cannot return an ABL array of a type that maps to the .NET array type (for example, INTEGER EXTENT) or the AVM raises a run-time error. If you do not execute any RETURN statement for *return-type* in the *method-logic*, the method returns the Unknown value (?) as its return value.

If you are defining an instance method with the OVERRIDE option, you can also call the overridden implementation of this method in the class hierarchy using the SUPER system reference. You can also optionally call any instance method defined in the same class using the THIS-OBJECT system reference.

If you are defining a static method, you cannot access any instance members of a class (including the defining class) nor can you use the SUPER and THIS-OBJECT system references. From a static method, you can access only other static members of a class and the local variables or other local data elements of the method.

Each logic statement must end with a period.

catch-block

Specifies a CATCH statement that defines error handling code for one or more error types. A DO block does not have any default error handling. Therefore, a DO block must have error handling options specified such that it becomes an undo-able block. Otherwise, ABL generates a compiler warning. For more information on *catch-block*, see the CATCH statement reference entry.

finally-block

Specifies a FINALLY statement that defines the processing that must occur after all other processing in the block occurs. For more information on *finally-block*, see the FINALLY statement reference entry.

END [METHOD]

Specifies the end of the method body definition. You must end the method body definition with the END statement.

Examples

The following example shows the definition of an instance method in a class (which might implement a method prototype declared in an interface, as depicted in the second example):

```
METHOD PUBLIC CHARACTER GetCustomerName (INPUT inCustNum AS INTEGER):

  FIND ttCust WHERE ttCust.CustNum = inCustNum NO-ERROR.
  IF AVAILABLE ttCust THEN
    RETURN ttCust.CustName.
  ELSE
    RETURN ?.

END METHOD.
```

The following example shows the definition of a method prototype declaration in an interface (which can be implemented by an instance method definition in a class, as depicted in the first example):

```
INTERFACE acme.myObjs.Interfaces.ICustObj:

  METHOD PUBLIC CHARACTER GetCustomerName (INPUT inCustNum AS INTEGER).

END INTERFACE.
```

For more examples of method definitions, including static and abstract methods, see the descriptions of `r-CustObj.cls`, `r-CustObjStatic.cls`, and `r-CustObjAbstract.cls` in the CLASS statement reference entry.

Notes

- You can terminate a METHOD statement with either a period (.) or a colon (:), but typically use a colon (:) for a method definition (in a class) and a period (.) for a method prototype (in an interface).
- A complete method definition must begin with the METHOD statement and end with the END statement.
- A method of a class can invoke another method of a class (instance or static, as appropriate), as well as an internal or external procedure or a user-defined function. Similarly, a procedure or user-defined function

can invoke a method of a class. Note that if *method-name* is an ABL reserved keyword, you might need to use the THIS-OBJECT system reference or Type-name syntax to call the method. For more information on referencing and invoking methods, see the reference entry for a Class-based method call.

- Local variables and other data elements defined within a method are scoped to the end of the method definition. The values of local variables and data elements do not persist across method invocations; they are re-initialized each time you invoke the method. However, if you define a local variable within a method using the same name as a data member or property within the class hierarchy, the local variable takes precedence over the data member or property for the duration of the method.
- When defining local variables and other data elements for any method, even one that overrides or implements a .NET method, you cannot define these elements using an AS data type (see Table 25: Explicit mappings between ABL and .NET data types). You must use an ABL data type like any other method definition. The ABL data that you assign to or from a given .NET method parameter must be assignment compatible with the .NET type, as in all other contexts.
- You cannot specify an access mode (for example, PUBLIC) for variable definitions in a method.
- You cannot define SHARED objects, work tables, temp-tables, or ProDataSet objects within the body of a method.
- ABL implements primitive and array parameters of methods as NO-UNDO variables.
- You can handle application errors in a method as in any ABL block. By executing a RETURN ERROR action at the block level or a THROW action at the block level with the presence of a ROUTINE-LEVEL ON ERROR UNDO, THROW statement, the AVM returns the ERROR condition to the statement that invoked the method. If a RETURN ERROR also includes the option to return a character string value, or you set the `ReturnValue` property of a `Progess.Lang.AppError` object that you THROW, you can get this value using the RETURN-VALUE function following the statement that invoked the method or in a CATCH block that catches the `Progress.Lang.AppError` object. For more information, see *Develop Object-oriented ABL Applications*.
- You **cannot** override an inherited .NET generic method. However, you can create a non-generic ABL method of the same name (without a type parameter list).
- You **cannot** override the following .NET methods:
  - `DestroyHandle( )`
  - `Dispose( )`
  - `Finalize( )`
  - `GetHashCode( )`

  In .NET, these methods are overridable. However, OpenEdge defines these methods as FINAL for an ABL session. Instead of overriding .NET methods (such as `Dispose( )`) that destroy or otherwise clean-up .NET resources, always use a destructor in the ABL class to manage class clean-up using these methods, even when inheriting from a .NET class. Similarly, if you want to return an ABL-processed value generated by `GetHashCode( )`, create a separate ABL method that calls `GetHashCode( )` to return the value.
- When you define a method in an ABL class that inherits a .NET class, you **cannot** define *method-name* as any of the following reserved method names (case insensitive) or ABL raises a compile-time error:
  - **`Get_`*property-name*`( )`** — Where *property-name* is the name of a property (including any default indexed property) defined by the .NET super class

    Note: For default indexed properties, *property-name* is usually `Item`.
  - **`Set_`*property-name*`( )`** — Where *property-name* is the name of a property (including any default indexed property) defined by the .NET super class

  - **Add_*event-name*( )** — Where *event-name* is the name of an event defined by the .NET super class
  - **Remove_*event-name*( )** — Where *event-name* is the name of an event defined by the .NET super class

- When you raise error from within an ABL method that overrides a .NET method or implements a .NET method defined in a .NET interface, if the method is called from .NET, and ABL error options raise the error out of the method block, ABL returns a .NET `System.Exception` to the caller. If the error is raised by executing a RETURN ERROR with the optional error string, the `Message` property of the `System.Exception` describes the operation where the error occurred, but the error string is available only to the ABL session, using the RETURN-VALUE function. If the error is raised by executing a RETURN ERROR with an optional ABL error object or by executing an UNDO, THROW, the `System.Exception`:`Message` property includes both a description of the operation where the error occurred and any messages from the ABL error object. If the error is fatal, the AVM responds as for any ABL class, generating a protrace file and exiting the session.

See also

Class-based method call, CLASS statement, FUNCTION statement, INTERFACE statement, Parameter definition syntax, PROCEDURE statement, RETURN statement, Type-name syntax, USING statement

---

MINIMUM function

Compares two or more values and returns the smallest.

Syntax

```
MINIMUM ( expression , expression [ , expression ] ... )
```

expression

A constant, field name, variable name, or expression. If there is a mixture of decimal and integer data types, decimal type is returned.

Example

This procedure prompts the user for an item number and how many of the item they want. If the number of items a user wants (stored in the want variable) is the minimum of the want variable and the `OnHand` field, the procedure displays an "enough in stock" message. Otherwise, the procedure displays a "not enough in stock" message.

r-minmum.p

```
DEFINE VARIABLE want NO-UNDO LIKE on-hand LABEL "How many do you want?".
DEFINE VARIABLE ans  AS LOGICAL NO-UNDO.

REPEAT:
  PROMPT-FOR Item.ItemNum want.
  FIND Item NO-LOCK USING Item.ItemNum.
  ans = FALSE.
  IF MINIMUM(INPUT want, Item.OnHand) = INPUT want THEN DO:
    MESSAGE "We have enough" Item.ItemName "in stock.".
    MESSAGE "Any other items to check?" UPDATE ans.
    IF NOT ans THEN LEAVE.
  END.
  ELSE DO:
    MESSAGE "We only have" Item.OnHand Item.ItemName "in stock.".
    MESSAGE "Any other items to check?"
    UPDATE ans.
    IF NOT ans THEN LEAVE.
  END.
END.
```

Notes

- When comparing character values, if at least one of the character fields is defined as case sensitive, then MINIMUM treats all of the values as case sensitive for the sake of the comparisons. If none of the values is case sensitive, MINIMUM treats lowercase letters as if they were uppercase letters.
- You cannot compare data of different DATE, DATETIME, and DATETIME-TZ data types to each other using MINIMUM. You must first convert different date and datetime data types to the same data type before doing a comparison between them.

See also

MAXIMUM function

MODULO operator

Determines the remainder after division.

Syntax

```
expression MODULO base
```

expression

An integer expression. If the expression could result in a decimal value, then that value will be rounded before the MODULO operation occurs. This may or may not produce a result that is sensible for your business logic. For code readability, such an expression should be explicitly converted to an integer expression with attention to whether or not a rounded or truncated integer is correct for your application. See the INTEGER, ROUND, and TRUNCATE functions

base

A positive integer expression that is the modulo base. For example, angles measured in degrees use a base of 360 for modulo arithmetic. 372 MODULO 360 is 12.

Example

This procedure determines the number of trucks required to ship a given quantity of material, and how much material is left over from a less than full truck load:

r-modulo.p

```
REPEAT:
 SET qty-avail AS INTEGER LABEL "Qty. Avail.".
 SET std-cap AS INTEGER LABEL "Std. Truck Capacity".
 DISPLAY TRUNCATE(qty-avail / std-cap,0) FORMAT ">,>>9" LABEL "# Full Loads"
   qty-avail MODULO std-cap LABEL "Qty. Left".
END.
```

Note

The *expression* must be greater than 0 for MODULO to return a correct value.

See also

INTEGER function, ROUND function, TRUNCATE function

MONTH function

Evaluates a date expression and returns a month INTEGER value from 1 to 12, inclusive.

Syntax

```
MONTH ( date )
```

```
MONTH ( datetime-expression )
```

date

A date expression where you want a month value.

datetime-expression

An expression that evaluates to a DATETIME or DATETIME-TZ. The MONTH function returns the month of the date part of the DATETIME or DATETIME-TZ value.

Example

This procedure displays all the Orders that have a `PromiseDate` in a month that has passed, and whose `ShipDate` field is the Unknown value (?), which is the initial value of the `ShipDate` field:

r-mon.p

```
FOR EACH Order NO-LOCK:
  IF (MONTH(Order.PromiseDate) < MONTH(TODAY) OR
    YEAR(Order.PromiseDate) < YEAR(TODAY)) AND Order.ShipDate = ? THEN
    DISPLAY Order.OrderNum LABEL "Order Num" Order.PO LABEL "P.O. Num"
      Order.PromiseDate LABEL "Promised By"
      Order.OrderDate LABEL "Ordered" terms
      WITH TITLE "These orders are overdue".
END.
```

See also

ADD-INTERVAL function, DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, ETIME function, INTERVAL function, ISO-DATE function, MTIME function, NOW function, TIME function, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

MTIME function

Returns an INTEGER value representing the time in milliseconds. If the MTIME function has no arguments, it returns the current number of milliseconds since midnight (similar to TIME, which returns seconds since midnight).

Syntax

```
MTIME ( [datetime-expression] )
```

datetime-expression

An expression that evaluates to a DATETIME or DATETIME-TZ. The MTIME function returns the time portion of *datetime-expression* in milliseconds.

If *datetime-expression* is a DATETIME-TZ, the MTIME function returns the local time relative to the time zone of the DATETIME-TZ value. For Example, a DATETIME-TZ field, *fdt*, is created in London (time zone UTC+00:00) with a value of `May 5, 2002 at 7:15:03.002 am`. MTIME(*fdt*) returns `26,103,002`, regardless of the session's time zone.

The MTIME function gets the current system time of the client or server machine that serves as the time source for applications running during the ABL session (specified by the TIME-SOURCE attribute).

See also

ADD-INTERVAL function, DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, ETIME function, INTERVAL function, ISO-DATE function, MONTH function, NOW function, TIME function, TIME-SOURCE attribute, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

NE or <> operator

Compares two expressions and returns a TRUE value if they are not equal.

Syntax

```
expression { NE | <> } expression
```

expression

A constant, field name, variable name, or expression. The expressions on either side of the NE or must be of the same data type.

Example

This procedure displays information for all items that appear in the catalog. (The `CatPage` field is not equal to the Unknown value (?) or 0.)

r-ne.p

```
FOR EACH Item NO-LOCK WHERE Item.CatPage <> ? AND Item.CatPage <> 0:
  DISPLAY Item.ItemNum Item.ItemName Item.CatPage
    WITH TITLE "Catalog Items" USE-TEXT.
END.
```

Notes

- By default, the AVM uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (`-cpcoll`) startup parameter take precedence over a collation specified for any database the AVM accesses during the session, except when the AVM uses or modifies pre-existing indexes. If you do not specify a collation with the `-cpcoll` startup parameter, the AVM uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, the AVM uses the collation rules with the default name "`basic`" (which might or might not exist in the `convmap.cp` file).
- If one of the expressions has the Unknown value (?) and the other does not, the result is TRUE. If both have the Unknown value (?), the result is FALSE. For SQL, however, if one or both expressions have the Unknown value (?), then the result is the Unknown value (?).
- You can compare character strings with NE. Most character comparisons are case insensitive in ABL. That is, all characters are converted to uppercase prior to comparisons. However, it is possible to define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If either *expression* is a field or variable defined as case sensitive, the comparison is case sensitive and "Smith" does not equal "smith".
- Characters are converted to their sort code values for comparison. Using the default case-sensitive collation table, all uppercase letters sort before all lowercase letters (for example, a is greater than Z, but less than b.) Note also that in character code uppercase A is less than [, \ , ^ , _, and ` , but lowercase a is greater than these.
- You cannot compare data of different DATE, DATETIME, and DATETIME-TZ data types to each other using NE. You must first convert different date and datetime data types to the same data type before doing a comparison between them.

- You can use NE to compare one BLOB field to another. The AVM performs a byte-by-byte comparison.
- You can use NE to compare a LONGCHAR variable to another LONGCHAR or CHARACTER variable. The variable values are converted to `-cpinternal` for comparison and must convert without error, or the AVM raises a run-time error.
- You can use NE to compare a CLOB field only to the Unknown value (?).
- You can use NE to compare two enums, including both ABL and .NET enums, as long as they are the same enum type. NE compares the underlying numerical values of the enumerators.

NEW function (classes)

Creates an instance of a class (object) and returns an object reference to that instance. You can use this object reference to access the `PUBLIC` data members, properties, and methods of the instance. For more information on object references, see the reference entry for a Class-based object reference.

Syntax

```
NEW object-type-name ( [ parameter[ , parameter ]...] )
```

object-type-name

Specifies the type name of the ABL or .NET class you want to instantiate. Specify a class type name using the syntax as described in the Type-name syntax reference entry. If *object-type-name* specifies a .NET object, it can be most any .NET class, with a few restrictions. For more information, see the Notes of this reference entry. The value of *object-type-name* is restricted to alphanumeric characters plus the symbols #, $, %, and _.

If, without casting, you assign the `NEW` function value to a target data element, or pass the `NEW` function as input to a target routine parameter, *object-type-name* must specify one of the following classes:

- The same class type that is defined for the target
- A subclass of the class type that is defined for the target
- A class type that implements the interface type defined for the target

If *object-type-name* specifies one of the following kinds of object types, ABL raises a compiler error:

- An interface type
- An abstract class type

([*parameter*[, *parameter*]...])

Specifies zero or more parameters passed to a `PUBLIC` instance constructor that is defined for the class. You must provide the parameters identified by the specified constructor, matched with respect to number, data type, and mode. To invoke a constructor that is overloaded in the class, you must specify sufficient information for each *parameter* to disambiguate it from all other constructors in the class. Otherwise, ABL raises an error identifying the ambiguity.

For information on the parameter passing syntax and disambiguating overloaded constructors, see the Parameter passing syntax reference entry.

For information on defining a constructor for a class, see the CONSTRUCTOR statement reference entry.

Examples

The following example shows how the NEW function can instantiate a class within an expression:

```
DEFINE VARIABLE rObj AS Progress.Lang.Object.

IF ( NEW Progress.Lang.Object( ) ):ToString( ) BEGINS "Progress" THEN
  MESSAGE "This object is making Progress..." VIEW-AS ALERT-BOX.
```

This code fragment instantiates a Progress.Lang.Object in order to check if the string value returned from its ToString( ) method begins with "Progress", and displays a message if it does.

The following example shows three (3) invocations of the NEW function.

r-newclass.p

```
ROUTINE-LEVEL ON ERROR UNDO, THROW.

FUNCTION ClassFunc RETURNS CLASS Progress.Lang.Object
  (INPUT iVal AS INTEGER):
  MESSAGE "iVal = " iVal VIEW-AS ALERT-BOX.
  IF iVal MODULO 2 = 0 THEN
    RETURN NEW Progress.Lang.Object( ).
  ELSE
    RETURN?.
END FUNCTION.

RUN DisplayClass( NEW Progress.Lang.Object( ) ).

PROCEDURE DisplayClass:
  DEFINE INPUT PARAMETER rObj AS CLASS Progress.Lang.Object.

  DEFINE VARIABLE cClass1 AS CHARACTER NO-UNDO.
  DEFINE VARIABLE cClass2 AS CHARACTER NO-UNDO.

  cClass1 = rObj:ToString( ).
  /* The ToString( ) call errors on an odd RANDOM input value */
  cClass2 = ( ClassFunc( RANDOM(1,100) ) ):ToString( ).

  MESSAGE "The " cClass1 " and " cClass2 " instances have the same
  class type."
    VIEW-AS ALERT-BOX.

  CATCH e AS Progress.Lang.SysError:
    UNDO, THROW NEW Progress.Lang.AppError( "Application Error", 555 ).
  END CATCH.
END PROCEDURE.
```

This r-newclass.p procedure runs an internal procedure (DisplayClass) that displays a message showing the ToString( ) values of two different objects instantiated as the same class type (Progress.Lang.Object). However, one of these objects is instantiated within a user-defined function (ClassFunc) that returns a valid object reference or the Unknown value (?) in order to generate an error, depending on the input value of a RANDOM function.

When a valid object reference is returned, it is used to access the ToString( ) method of the instance in an expression. When the Unknown value (?) is returned, this generates a run-time error when used as an object reference. The procedure then catches the error and responds by throwing a Progress.Lang.AppError object that is also instantiated by a NEW function invoked in an expression, and the error text is displayed as an error message when run in the OpenEdge Editor.

Notes

- When you create an instance of a class, ABL invokes the specified constructor for the class. At this time, the object instance gets its own copy of the data members and properties defined in the class and in all classes within its inherited class hierarchy. In addition, the object instance is added to the session object chain referenced by the FIRST-OBJECT attribute or LAST-OBJECT attribute of the SESSION system handle.
- The ABL Virtual Machine (AVM) automatically deletes (garbage collects) any class instance that you create with the NEW function some time after no reference to that object exists in the ABL session. However, you can force any class instance to be deleted immediately by using the DELETE OBJECT statement. For more information on garbage collection for class instances, see the DELETE OBJECT statement reference entry.
- You can assign the object reference value returned by the NEW function to an ABL data element (or target of a passed parameter) defined as a class type or interface type when the destination data element (or parameter target) is defined as the same class as, as a super class of, or as an interface implemented by the instantiated class. In any case, the target data element retains its defined class or interface type. For more information on assigning object references to ABL data elements, see the NEW statement and Assignment (=) statement reference entries.
- You can use an object reference as a parameter or return type for methods, internal and external procedures, and user-defined functions (see the example).
- You can pass an object reference as a parameter between an application server and an ABL client, although there are restrictions on the types of object you can pass. See the Parameter definition syntax entry for the full list of restrictions.
- You can pass a temp-table or ProDataSet containing an object reference field as a parameter between an application server and an ABL client, as long as the objects themselves satisfy the restrictions described in the Parameter definition syntax entry.
- You can only access PRIVATE class members using an object reference in instances of the class where the members are defined. You can only access PACKAGE-PRIVATE class members using an object reference in an instance of the class, or any class within its package, where the member is defined. You can only access PROTECTED class members using an object reference in an instance of the class or subclass where the member is defined. You can only access PACKAGE-PROTECTED class members using an object reference in an instance of the class, any class within its package, and from within any subclass that inherits the class, where the member is defined. To allow access outside of these situations, you must provide PUBLIC methods or properties to do so. For information on accessing data members, properties, or methods using an object reference, see the Class-based object reference entry.
- Class-based object instances are not associated with handle-based objects, such as socket or procedure objects, and object references are not compatible with object handles. Thus, for example, you cannot use an object reference in any statement or function that returns a value of type HANDLE, and you cannot pass a procedure handle to a method that expects an object reference.
- You can compare two object references for equality using the EQ (=) operator, which determines if two object references are referencing the same object instance. Two object references can be equal even when you define their data types for different classes in the same class hierarchy as long as they each point to the same class instance. So, if ClassA is a super class of ClassB, and two object references point to the same ClassB instance, if one object reference is defined with the ClassA data type and the other object reference is defined with the ClassB data type, the two object references are equal. However, if the ClassA object reference points to a different class instance than the ClassB object reference, the two object

references are not equal. Two object references are also equal if they are both set to the Unknown value (?).

- You can use the `Equals( )` method in the `Progress.Lang.Object` class to compare the data members and properties of two object references, as long as this class provides an implementation of the `Equals( )` method.
- If *`object-type-name`* specifies a .NET object, note that in ABL you cannot instantiate the following .NET classes:
  - Any .NET class that is defined in the default namespace, that is, where the class name is the complete object type name
  - `System.Threading.Thread` or any class derived from it
  - `System.Delegate` or any delegate type derived from it
- If *`object-type-name`* specifies a GUI or non-GUI .NET object type, you **can** instantiate it within a non-GUI ABL session on Windows, including a:
  - Character mode (CHUI) client
  - Batch-mode client
  - Progress Application Server (PAS) for OpenEdge instance

  However, you **cannot** block for any .NET object events or visualize any GUI objects in a non-GUI ABL session.
- This function can raise errors during the execution of constructors for the class being instantiated, or for any class in its inherited class hierarchy. For example:
  - A constructor in the class hierarchy executes the RETURN statement with the ERROR option or the UNDO statement with the `THROW` or `RETURN ERROR` options.
  - The class definition file for the class, a super class, or an interface could not be found.
  - The run-time parameters of the constructor for the class, or a constructor for a class in the inherited class hierarchy, are not compatible.

  When the AVM encounters one of these errors, and the constructor cannot create the class instance or its inherited class hierarchy, the AVM automatically invokes the destructor for any class that has already been constructed while building the class hierarchy for the object.

  For more information on errors raised by instantiating classes, see *Develop Object-oriented ABL Applications*.
- Any errors returned during class instantiation, including application errors returned by constructors executing `RETURN ERROR` or `UNDO, THROW`, are handled by the statement that invokes the NEW function. In any case, if a `RETURN ERROR` in the constructor of the instantiating class also returns an optional character string value, this value is available using the `RETURN-VALUE` function following the statement where this `NEW` function appears.
- You can use the `NEW` function in the `VAR` statement.

See also

Assignment (=) statement, Class-based object reference, CLASS statement, CONSTRUCTOR statement, DELETE OBJECT statement, DYNAMIC-NEW statement, FIRST-OBJECT attribute, FUNCTION statement, LAST-OBJECT attribute, METHOD statement, New( ) method, NEW statement, Parameter passing syntax, Type-name syntax, USING statement , VAR statement

NEW function (record buffers)

Checks a record buffer and returns a TRUE value if the record in that buffer is newly created. If the record was read from the database, NEW returns a FALSE value.

Syntax

```
NEW record
```

record

The name of the record buffer you want to check with the NEW function.

To use the NEW function with a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

Example

This procedure enters new Orders, optionally creating a Customer record if one does not exist. The NEW function is later used to select alternate processing depending if a Customer is newly created or already exists.

r-newrecord.p

```
REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum NO-ERROR.
  IF NOT AVAILABLE Customer THEN
  DO WITH FRAME newcus:
    MESSAGE "Creating new customer".
    CREATE Customer.
    ASSIGN Customer.CustNum.
    UPDATE Customer.Name Customer.Address Customer.City Customer.State
      Customer.Country.
  END.

  CREATE Order.
  Order.CustNum = Customer.CustNum.
  IF NEW Customer THEN DO:
    UPDATE Order.OrderNum Order.PromiseDate.
    Order.Terms = "COD".
    DISPLAY Order.Terms.
  END.
  ELSE
    UPDATE Order.OrderNum Order.PromiseDate Order.Terms.
END.
```

Note

The NEW function returns a TRUE value only during the transaction in which the record is created. If the scope of the record is greater than the transaction in which the record is created, the NEW function returns a FALSE value outside the transaction.

See also

AVAILABLE function, FIND statement, LOCKED function, Record phrase

NEW statement

Creates an instance of a class (object) using the NEW function and assigns its object reference to an appropriately defined ABL data element. Once assigned, you can use the object reference to access this class instance and its PUBLIC data members, properties, and methods. For more information on object references, see the reference entry for a Class-based object reference:

Syntax

```
object-reference = new-function [ NO-ERROR ]
```

object-reference

The name of an ABL data element to which you want to assign the object reference of a new instance of the class specified by *new-function*. This data element must be defined as a compatible class or interface type and can be one of the following:

- A temp-table field defined as a Progress.Lang.Object class type using the DEFINE TEMP-TABLE statement
- A variable scoped to the current procedure, user-defined function, or method of a class or an accessible class-based variable data member defined as a class or interface type using the VAR statement or DEFINE VARIABLE statement
- A parameter defined as a class or interface type for the current procedure, user-defined function, or method of a class using an appropriate DEFINE PARAMETER statement or Parameter definition syntax
- An accessible and writable class-based property defined as a class or interface type using the DEFINE PROPERTY statement

To be compatible, the object type of *object-reference* must be:

- The same class type as the class instantiated by *new-function*
- A super class of the class instantiated by *new-function*
- An interface that is implemented by the class instantiated by *new-function*

new-function

An invocation of the NEW function, which creates an instance of a specified class and returns an object reference to that instance. This is the syntax for the NEW function, where *object-type-name* must specify a class type consistent with the object type of *object-reference*: The value of *new-function* is restricted to alphanumeric characters plus the symbols #, $, %, and _.

```
NEW object-type-name ( [ parameter[ , parameter ]...] )
```

The *parameter* list specifies the class constructor that is used to instantiate the class. For more information on the syntax and operation of the NEW function, see the NEW function (classes) reference entry.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

For the NEW statement with NO-ERROR, if ERROR is raised, then the *object-reference* remains unchanged.

Example

The following code fragment shows the definition of a variable that is assigned to the object reference for a new class instance:

```
DEFINE VARIABLE myCustObj AS CLASS acme.myObjs.CustObj NO-UNDO.

myCustObj = NEW acme.myObjs.CustObj ( ).
```

Notes

- After the assignment, *object-reference* contains a copy of the object reference value returned by *new-function*, which points to the same object instance, not a copy of the object created by *new-function*.
- Although you can assign an object reference to a temp-table field defined as a Progress.Lang.Object class type, you cannot assign an object reference to a field in a database table. For more information, see *Develop Object-oriented ABL Applications*.
- This statement can raise errors during the execution of constructors for the class being instantiated by *new-function*, or for any class in its inherited class hierarchy. For example:
  - A constructor in the class hierarchy executes the RETURN statement with the ERROR option or the UNDO statement with the THROW or RETURN ERROR options.
  - The class definition file for the class, a super class, or an interface could not be found.
  - The run-time parameters of the constructor for the class, or a constructor for a class in the inherited class hierarchy, are not compatible.

  When the AVM encounters one of these errors, and the constructor cannot create the class instance or its inherited class hierarchy, the AVM automatically invokes the destructor for any class that has already been constructed while building the class hierarchy for the object.

  For more information on errors raised by instantiating classes, see *Develop Object-oriented ABL Applications*.
- The ABL Virtual Machine (AVM) automatically deletes (garbage collects) any class instance that you create with the NEW statement some time after no reference to that object exists in the ABL session. However, you can force any class instance to be deleted immediately by using the DELETE OBJECT statement. For more information on garbage collection for class instances, see the DELETE OBJECT statement reference entry.

See also

Assignment (=) statement, CAST function, Class-based object reference, CLASS statement, DYNAMIC-NEW statement, NEW function (classes), New( ) method, NO-ERROR option, Parameter passing syntax , VAR statement

NEXT statement

Goes directly to the END of an iterating block and starts the next iteration of the block.

Syntax

```
NEXT [ label ]
```

label

The name of the block for which you want to start the next iteration. If you do not name a block, the AVM starts the next iteration of the innermost iterating block that contains the NEXT statement.

Example

The FOR EACH block in this procedure reads a single Customer record on each iteration of the block. If the SalesRep field of a Customer record does not match the SalesRep value supplied to the PROMPT-FOR statement, the NEXT statement causes the AVM to do the next iteration of the FOR EACH block, bypassing the DISPLAY statement.

r-next.p

```
PROMPT-FOR Customer.SalesRep LABEL "Enter salesman initials"
  WITH SIDE-LABELS CENTERED.
FOR EACH Customer:
  IF Customer.SalesRep <> INPUT Customer.SalesRep THEN NEXT.
  DISPLAY Customer.CustNum Customer.Name Customer.City Customer.State
    WITH CENTERED USE-TEXT.
END.
```

See also

LEAVE statement

NEXT-PROMPT statement

Specifies the field in which you want to position the cursor during the next input operation that involves that field in a frame.

Note: Does not apply to SpeedScript programming.

Syntax

```
NEXT-PROMPT field [ frame-phrase ]
```

field

Indicates the name of the input field in which you want to place the cursor the next time the user supplies input to the frame. If the field you name is not an input field in the frame, the AVM disregards the NEXT-PROMPT statement.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see theFrame phrase reference entry.

Example

This procedure lets you update Customer information. If you do not enter a value for Customer.Contact, the AVM positions the cursor in the Contact field when the UPDATE statement is processed following the UNDO, RETRY of the FOR EACH block.

r-nprmpt.p

```
FOR EACH Customer:
  UPDATE Customer WITH 2 COLUMNS.
  IF Customer.Contact EQ " " THEN DO:
    MESSAGE "You must enter a contact".
    NEXT-PROMPT Customer.Contact.
    UNDO, RETRY.
  END.
END.
```

Notes

- NEXT-PROMPT is useful in an EDITING phrase because it can dynamically reposition the cursor depending on input from the user.
- When you have to do complex field checking that you are unable to do in a Dictionary validation expression or in a VALIDATE option of the Frame phrase, use NEXT-PROMPT to position the cursor after detecting an error.
- If the next data entry statement involving the frame specified with NEXT-PROMPT does not use the indicated NEXT-PROMPT field, then the AVM ignores the NEXT-PROMPT statement.
- The NEXT-PROMPT statement can affect default frame layout. In this procedure, the AVM prompts for a and b (in that order):

  r-nextp.p

  ```
  DEFINE VARIABLE a AS CHARACTER NO-UNDO.
  DEFINE VARIABLE b AS CHARACTER NO-UNDO.

  UPDATE a b.
  ```

  However, if you include NEXT-PROMPT b before the update statement, as shown in the following procedure, the AVM prompts for b first and a second:

r-nextp1.p

```
DEFINE VARIABLE a AS CHARACTER NO-UNDO.
DEFINE VARIABLE b AS CHARACTER NO-UNDO.

NEXT-PROMPT b.
UPDATE a b.
```

See also

EDITING phrase, Frame phrase

NEXT-VALUE function

Returns the next INT64 value of a static sequence, incremented by the positive or negative value defined in the Data Dictionary.

Syntax

```
NEXT-VALUE ( sequence [ , logical-dbname ] [ , tenant-id ] )
```

sequence

An identifier that specifies the name of a sequence defined in the Data Dictionary.

logical-dbname

An identifier that specifies the logical name of the database in which the sequence is defined. The database must be connected. If multiple databases are connected, you can omit this parameter if you specify a sequence that is unique to one of the databases.

tenant-id

An integer expression that evaluates to the tenant ID of a regular tenant, including the default tenant (`0`). This option applies only to a multi-tenant sequence specified by *sequence* and is intended for access primarily by a super-tenant user.

If you are a regular-tenant user and you specify *tenant-id*, the specified tenant must be the same as the tenant associated with the database connection identity for *logical-dbname* or the AVM raises a run-time error.

If you are a super-tenant user and you do not specify *tenant-id*, the function assumes the effective tenant ID (GET-EFFECTIVE-TENANT-ID function).

If the sequence specified by *sequence* is non-multi-tenant (a shared sequence in a multi-tenant database), and you specify *tenant-id*, the option is ignored.

Example

The following trigger procedure uses the NextItemNum sequence to set the `ItemNum` field for a new Item record:

r-critem.p

```
TRIGGER PROCEDURE FOR Create OF Item.

/* Automatically assign a unique item number using NextItemNum seq */
ASSIGN Item.ItemNum = NEXT-VALUE(NextItemNum).
```

Notes

- If *sequence* is a cycling sequence, and the NEXT-VALUE function increments the sequence beyond its upper limit (for positive increments) or decrements the sequence beyond its lower limit (for negative increments), the function sets and returns the initial value defined for the sequence.
- If *sequence* is a terminating sequence, and the NEXT-VALUE function attempts to increment the sequence beyond its upper limit (for positive increments) or decrement the sequence beyond its lower limit (for negative increments), the function returns the Unknown value (?) and leaves the current sequence value unchanged. Once a sequence terminates, NEXT-VALUE continues to return the Unknown value (?) for the specified sequence until it is reset to a new value with the CURRENT-VALUE statement, or its definition is changed to a cycling sequence. After changing the sequence definition to cycle, the first use of NEXT-VALUE for the sequence sets and returns its initial value.
- If *sequence* is a multi-tenant sequence in the database, each regular tenant has their own current value of the sequence. So, the same values are returned for each tenant that invokes this function. If the sequence is shared in a multi-tenant database, the values returned by this function are unique across all tenants in the database.

  Caution: Be very careful using multi-tenant sequences when you are updating shared tables or multi-tenant tables for a tenant group. For more information, see the sections on using multi-tenant-enabled sequences in *OpenEdge Programming Interfaces*.

- The value of a sequence set by the NEXT-VALUE function persists in the database until the next CURRENT-VALUE statement or NEXT-VALUE function is invoked for the sequence, or until the sequence is deleted from the database.
- You cannot invoke the NEXT-VALUE function from within a WHERE clause. Doing so generates a compiler error because the value returned by the NEXT-VALUE function can result in ambiguous expressions. To use a result from the NEXT-VALUE function in a WHERE clause, assign the result to a variable and use the variable in the WHERE clause instead.
- You can use any combination of the NEXT-VALUE function, CURRENT-VALUE function, CURRENT-VALUE statement, and their dynamic versions. Use the dynamic version when you don't know what the database name or sequence name is at compile time.
- Be careful when accessing a database sequence with an alias that points to a different database than the one used when the alias was defined. If you supply an alias name to the CURRENT-VALUE function or the NEXT-VALUE function, only the database used to define the alias is referenced. In this case, it is preferable to use the DYNAMIC-CURRENT-VALUE function and DYNAMIC-NEXT-VALUE function instead of the CURRENT-VALUE function and NEXT-VALUE function, respectively.

See also

CURRENT-VALUE function, CURRENT-VALUE statement, DYNAMIC-CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, DYNAMIC-NEXT-VALUE function, NEXT-VALUE function

NORMALIZE function

Returns the normalized form of a character string based on the specified Unicode normalization form.

Syntax

```
NORMALIZE ( string , normalization-form )
```

string

The source string to normalize. The value may be of type CHARACTER or LONGCHAR.

If the string is a CHARACTER value, `-cpinternal` must be set to UTF-8. If the string is a LONGCHAR value, its code page can be any form of Unicode (for example, UTF-8, UTF-16, or UTF-32). This function returns a value of the same data type as the source string.

normalization-form

A character expression that evaluates to one of the following Unicode normalization forms:

- **NFD** — Canonical Decomposition
- **NFC** — Canonical Decomposition, followed by Canonical Composition
- **NFKD** — Compatibility Decomposition
- **NFKC** — Compatibility Decomposition, followed by Canonical Composition
- **NONE** — Returns the source string unchanged

NOT operator (bitwise)

Performs a bitwise NOT operation on a flag enum instance and returns an instance of the same type.

Syntax

```
NOT flag-enum-expression
```

flag-enum-expression

An expression whose value is a flag enum instance. The expression cannot be the Unknown value (?).

Example

You can use bitwise NOT to turn off a flag. This code fragment defines a variable of type `Progres.Reflect.Flags` (see the Progress.Reflect.Flags enumeration entry for a full list of enumerators) and uses NOT to make sure the `Static` flag is not set while leaving the other flags unchanged.

```
DEFINE VARIABLE flagsEnum AS Progress.Reflect.Flags.

flagsEnum = flagsEnum AND NOT Flags:Static.
```

Notes

- This operator can be used with both ABL and .NET flag enums.

See also

AND operator (bitwise), ENUM statement, OR operator (bitwise), XOR operator (bitwise)

NOT operator (logical)

Returns TRUE if an expression is false, and FALSE if an expression is true.

Syntax

```
NOT expression
```

expression

A logical expression whose value is logical, that is TRUE/FALSE, YES/NO.

Example

In this procedure, if the user enters the number of a Customer that does not exist, the procedure displays a message that the Customer does not exist and the user must try again. If the Customer does exist, the procedure displays the `Name` and `Phone` number of the Customer.

r-not.p

```
REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum NO-ERROR.
  IF NOT AVAILABLE customer THEN DO:
    MESSAGE "Customer with CustNum:" INPUT Customer.CustNum
      " does not exist. Please try another.".
    UNDO, RETRY.
  END.
  ELSE
    DISPLAY Customer.Name Customer.Phone.
END.
```

See also

AND operator (logical), OR operator (logical)

NOT ENTERED function

Returns a TRUE value if a frame field was not modified during the last INSERT, PROMPT-FOR, SET, or UPDATE statement.

Note: Does not apply to SpeedScript programming.

Syntax

```
[ FRAME frame ] field NOT ENTERED
```

FRAME *frame*

The frame name that contains the field named by the *field* argument. If you do not name a frame, the NOT ENTERED function starts with the current frame and searches outward until it finds the field you name with the *field* argument.

field

The name of the field or variable you are checking.

Example

This procedure displays the `CustNum`, `Name`, and `CreditLimit` for each Customer. For each Customer, the procedure prompts the user for a new `CreditLimit` value. The NOT ENTERED function tests to see if you enter a value. If you enter a value and it is different from the present value of `CreditLimit`, the procedure displays the old and new `CreditLimit` values. If you enter the same value or no value, the procedure displays a message that the `CreditLimit` has not been changed.

r-nenter.p

```
DEFINE VARIABLE new-max NO-UNDO LIKE Customer.CreditLimit.

FOR EACH Customer:
  DISPLAY Customer.CustNum Customer.Name Customer.CreditLimit
    LABEL "current max credit"
    WITH FRAME a 1 DOWN ROW 1.
  SET new-max LABEL "new max credit"
    WITH SIDE-LABELS NO-BOX ROW 10 FRAME b.
  IF new-max NOT ENTERED OR new-max = Customer.CreditLimit THEN DO:
    DISPLAY "No Change In credit-limit" WITH FRAME d ROW 15.
    NEXT.
  END.
  DISPLAY "Changing Credit Limit of" Customer.Name SKIP
    "from" Customer.CreditLimit "to" new-max WITH FRAME c ROW 15 NO-LABELS.
  Customer.CreditLimit = new-max.
END.
```

Note

If you use a field or variable referenced with NOT ENTERED in more than one frame, then the AVM uses the value in the frame most recently introduced in the procedure. To make sure you are using the appropriate frame, use the FRAME option with the NOT ENTERED function to reference a particular frame.

See also

ENTERED function

NOW function

Returns the current system date, time, and time zone as a DATETIME-TZ value.

The NOW function returns the system date and time of the client or server machine that serves as the time source for applications running during the ABL session (specified by the TIME-SOURCE attribute).

Syntax

```
NOW
```

Example

Following is an example of using the NOW function:

```
DEFINE VARIABLE v-datetime    AS DATETIME    NO-UNDO.
DEFINE VARIABLE v-datetime-tz AS DATETIME-TZ NO-UNDO.

ASSIGN
  v-datetime    = NOW
  v-datetime-tz = NOW.
```

See also

ADD-INTERVAL function, DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, ETIME function, INTERVAL function, ISO-DATE function, MONTH function, MTIME function, TIME function, TIME-SOURCE attribute, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

NO-ERROR option

NO-ERROR

If an error occurs on the statement with NO-ERROR, the AVM undoes the action of the statement and continues execution with the next statement rather than raising ERROR. In addition, all error messages are diverted to the ERROR-STATUS system handle rather than being displayed to the current output device. The information stored in the ERROR-STATUS system handle remains until the next statement using NO-ERROR. To check for errors after a statement that uses the NO-ERROR option:

- Check the ERROR-STATUS:ERROR attribute to see if an error was encountered during the execution of the statement. Note however that sometimes errors are treated as warnings and ERROR-STATUS:ERROR is not set.
- Alternatively, check if the ERROR-STATUS:NUM-MESSAGES attribute is greater than zero to see if the AVM generated error or warning messages.
- Use ERROR-STATUS:GET-MESSAGE(*message-num*) to retrieve a particular message, where *message-num* is 1 for the first message.

NO-ERROR has no effect when a STOP or QUIT condition occurs.

If an error object is thrown to a statement that includes the NO-ERROR option, the information and messages in the error object are used to set the ERROR-STATUS system handle.

For more information on using NO-ERROR and error handling in general, see *ABL Error Handling*.

See also

Assignment (=) statement, ASSIGN statement, BUFFER-COMPARE statement, BUFFER-COPY statement, Class-based method call, COMPILE statement, CONNECT statement, COPY-LOB statement, CREATE automation object statement, CREATE CALL statement, CREATE DATABASE statement, CREATE SAX-ATTRIBUTES statement, CREATE SAX-READER statement, CREATE SAX-WRITER statement, CREATE SERVER-SOCKET statement, CREATE SOCKET statement, CREATE statement, CREATE WIDGET-POOL statement, DDE ADVISE statement, DDE EXECUTE statement, DDE GET statement, DDE INITIATE statement, DDE REQUEST statement, DDE SEND statement, DDE TERMINATE statement, DELETE OBJECT statement, DELETE PROCEDURE statement, DELETE statement, DELETE WIDGET-POOL statement, DISCONNECT statement, DISPLAY statement, DYNAMIC-NEW statement, EXTENT statement, FIND statement, IMPORT statement, EMPTY TEMP-TABLE statement, INSERT statement, LOAD statement, NEW statement, Publish( ) event method, PUT-KEY-VALUE statement, RAW-TRANSFER statement, RELEASE OBJECT statement, RELEASE statement, REPOSITION statement, RUN STORED-PROCEDURE statement, RUN SUPER statement, SAVE CACHE statement, SET statement, SUBSCRIBE statement, Subscribe( ) event method, SUPER system reference, UNLOAD statement, Unsubscribe( ) event method, UPDATE statement, USE statement, VALIDATE statement

NUM-ALIASES function

Returns an INTEGER value that represents the number of aliases defined. The NUM-ALIASES function uses no arguments.

Syntax

```
NUM-ALIASES
```

Example

This procedure displays the number of defined aliases. It also displays the aliases and logical database names of all connected databases.

r-numal.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

DISPLAY NUM-ALIASES LABEL "Number of Defined Aliases:".
REPEAT ix = 1 TO NUM-ALIASES:
  DISPLAY ALIAS(ix) LABEL "Aliases"
    LDBNAME(ALIAS(ix)) LABEL "Logical Database".
END.
```

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

NUM-DBS function

Takes no arguments; returns the number of connected databases as an INTEGER value.

Syntax

```
NUM-DBS
```

Example

This procedure uses NUM-DBS to display the logical name and database restrictions of all connected databases:

r-numdbs.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

REPEAT ix = 1 TO NUM-DBS:
  DISPLAY LDBNAME(ix) DBRESTRICTIONS(ix) FORMAT "x(40)".
END.
```

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, PDBNAME function, SDBNAME function

NUM-ENTRIES function

Returns the number of elements in a list of character strings as an INTEGER value.

Syntax

```
NUM-ENTRIES ( list [ , character ] )
```

list

A character expression containing a list of character strings separated with a character delimiter. The list can be a variable of type CHARACTER or LONGCHAR. NUM-ENTRIES returns the number of elements in the list. Specifically, NUM-ENTRIES returns the number of delimiters plus 1, and it returns 0 if *list* equals the empty string ("").

character

A delimiter you define for the list. The default is a comma (,). This allows functions to operate on non-comma-separated lists. If you use an alphabetic character, this delimiter is case sensitive.

Examples

This procedure uses NUM-ENTRIES and ENTRY to loop through a list of regions and display them, one per line. Since there are obviously five regions, the REPEAT statement, REPEAT ix = 1 TO 5, works fine here.

r-n-ent1.p

```
DEFINE VARIABLE ix      AS INTEGER   NO-UNDO.
DEFINE VARIABLE regions AS CHARACTER NO-UNDO
  INITIAL "Northeast,Southest,Midwest,Northwest,Southwest".

REPEAT ix = 1 TO NUM-ENTRIES(regions):
  DISPLAY ENTRY(ix, regions) FORMAT "x(12)".
END.
```

In the following example, PROPATH is a comma-separated list of unknown length:

r-n-ent2.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

REPEAT ix = 1 TO NUM-ENTRIES(PROPATH):
  DISPLAY ENTRY(ix, PROPATH) FORMAT "x(64)".
END.
```

This procedure uses NUM-ENTRIES to loop through the PROPATH (a comma-separated list of directory paths) and print the directories, one per line.

This example uses a list that does not use commas as a delimiter. This procedure returns a value of 13:

r-n-ent3.p

```
DEFINE VARIABLE sentence AS CHARACTER NO-UNDO INITIAL
  "This sentence would be seven words long if it were six words shorter".

DISPLAY NUM-ENTRIES(sentence," ").
```

Note

The NUM-ENTRIES function is multi-byte enabled. The specified *list* can contain entries that have multi-byte characters and the *character* delimiter can be a multi-byte character.

See also

ENTRY function, ENTRY statement, LOOKUP function

NUM-RESULTS function

Returns, as an INTEGER value, the number of rows currently in the results list of a scrolling query. The results list is initialized when the query is opened. Depending on the query, the entire list is built immediately upon opening or it is gradually as needed.

Syntax

```
NUM-RESULTS ( query-name )
```

query-name

A character expression that evaluates to the name of a currently open, scrolling query. If *query-name* does not resolve to the name of a query, or if the query is not open or not scrolling, then the function returns the Unknown value (?).

Note: Searching for a query using a handle is more efficient than a character expression. The AVM resolves a character expression at run time by searching in the current routine for a static query with that name. If not found, the AVM searches the enclosing main procedure. If still not found, the AVM searches up through the calling programs of the current routine, and their main procedures. Since a handle uniquely identifies a query, no such search is required. Use the query object handle's NUM-RESULTS attribute to avoid a run-time search.

Example

The following example uses the NUM-RESULTS function in a message to report on the number of rows in a browse. Note that the query is opened with the PRESELECT option so that the entire results list is built immediately. Otherwise, NUM-RESULTS might not return the total number of rows in the browse. When you run this procedure and choose a button, the AVM selects certain rows within the browse and then reports on the number of rows selected and the total number of rows in the browse.

r-brownr.p

```
DEFINE VARIABLE curr-rec  AS ROWID   NO-UNDO.
DEFINE VARIABLE status-ok AS LOGICAL NO-UNDO.
DEFINE VARIABLE threshold NO-UNDO LIKE Customer.CreditLimit INITIAL 25000.

DEFINE BUTTON no-orders-custs LABEL "No Orders".
DEFINE BUTTON hi-cred-custs   LABEL "High Credit".

DEFINE QUERY qry FOR Customer.
DEFINE BROWSE brws QUERY qry DISPLAY Customer.CustNum Customer.Name
  Customer.Country Customer.CreditLimit
  WITH 10 DOWN MULTIPLE.
```

```
FORM
  brws SKIP(1)
  no-orders-custs hi-cred-custs
  WITH FRAME brws-frame.

FORM threshold
  WITH FRAME thresh-frame VIEW-AS DIALOG-BOX TITLE "Set Threshold"
  SIDE-LABELS.

ON CHOOSE OF no-orders-custs DO:
  /* Select those customers with no orders. */
  status-ok = brws:DESELECT-ROWS().
  HIDE MESSAGE.
  FOR EACH Customer NO-LOCK WHERE NOT CAN-FIND(FIRST Order OF Customer):
    /* Position query to this record and then select row in browse. */
    curr-rec = ROWID(Customer).
    REPOSITION qry TO ROWID curr-rec.
    status-ok = brws:SELECT-FOCUSED-ROW().
    IF NOT status-ok THEN
      MESSAGE "Could not select row.".
  END.

  /* Report number of selected rows and position to first selected. */
  MESSAGE brws:NUM-SELECTED-ROWS "of" NUM-RESULTS("qry")
    "rows have been selected.".

  IF brws:NUM-SELECTED-ROWS > 0 THEN
    status-ok = brws:SCROLL-TO-SELECTED-ROW(1).
END. /* ON CHOOSE OF no-orders-cust */

ON CHOOSE OF hi-cred-custs DO:
  /* Select customers with high credit limits. */
  status-ok = brws:DESELECT-ROWS().
  HIDE MESSAGE.

  /* Get CreditLimit threshold value. */
  UPDATE threshold WITH FRAME thresh-frame.
  FOR EACH Customer NO-LOCK WHERE Customer.CreditLimit >= threshold:
    /* Position query to this record and then select row in browse. */
    curr-rec = ROWID(Customer).
    REPOSITION qry TO ROWID curr-rec.
    status-ok = brws:SELECT-FOCUSED-ROW().
    IF NOT status-ok THEN
      MESSAGE "Could not select row.".
  END. /* ON CHOOSE OF hi-cred-custs */

  /* Report number of selected rows and position to first selected. */
  MESSAGE brws:NUM-SELECTED-ROWS "of" NUM-RESULTS("qry")
    "rows have been selected.".

  IF brws:NUM-SELECTED-ROWS > 0 THEN
    status-ok = brws:SCROLL-TO-SELECTED-ROW(1).
END.

OPEN QUERY qry PRESELECT EACH Customer.

ENABLE ALL WITH FRAME brws-frame.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- To use the NUM-RESULTS function with a query, the query must be associated with a browse widget or you must define the query with the SCROLLING option. For more information on query definitions, see the reference entry for the DEFINE QUERY statement.
- If the query is empty, NUM-RESULTS returns 0.

- When possible, the AVM performs optimizations for GET LAST and REPOSITION statements. These optimizations make the results list invalid. At that point, NUM-RESULTS returns the Unknown value (?). These optimizations do not occur if the query is opened with the PRESELECT option or has an associated browse widget.

See also

CLOSE QUERY statement, CURRENT-RESULT-ROW function, DEFINE BROWSE statement, DEFINE QUERY statement, GET statement, NUM-RESULTS attribute, OPEN QUERY statement, QUERY-OFF-END function, REPOSITION statement

ON ENDKEY phrase

Defines the processing that occurs when the ENDKEY condition arises during a block. This condition usually occurs when the user presses **END-ERROR** during the first interaction of a block iteration, or any time the user presses a defined **END-KEY**.

If you use a REPEAT or FOR EACH block, the default processing for ENDKEY is to undo all the processing in the current iteration of the block, then leave the block and continue on to any remaining statements in the routine.

Note: Does not apply to SpeedScript programming.

Syntax

```
ON ENDKEY UNDO
  [ label1 ]
  [      , LEAVE [ label2 ]
       | , NEXT [ label2 ]
       | , RETRY [ label1 ]
       | , RETURN [ return-value |
                  ERROR [ return-value | error-object-expression ] |
                  NO-APPLY ]
  ]
```

label1

The name of the block whose processing you want to undo. If you do not name a block with *label1*, the AVM undoes the processing of the current block.

LEAVE [*label2*]

Indicates that, after undoing the processing of a block, the AVM leaves the block labeled *label2*. If you do not name a block, the AVM leaves the current block. There are restrictions. For example, you cannot undo an outer block, but leave only the inner block.

NEXT [*label2*]

Indicates that, after undoing the processing of a block, the AVM executes the next iteration of the block you name with the *label2* option. If you do not name a block with the NEXT option, the AVM executes the next iteration of the current block.

RETRY [*label1*]

Indicates that after undoing the processing of a block, the AVM repeats the same iteration of the block.

Because RETRY in a block without user input results in an infinite loop, the AVM automatically checks for this possibility and converts a RETRY block into a LEAVE block, or a NEXT block if it is an iterating block. This behavior is often referred to as infinite loop protection.

RETURN ...

Returns to the calling routine, if there is one. The following table describes various RETURN options:

Option	Description
return-value	In procedures and VOID methods, this must be a CHARACTER string. The caller can use the RETURN-VALUE function to read the returned value. For user-defined functions, non-VOID methods and property getters, the value must match the specified return type.
ERROR	Undoes the current subtransaction, and raises ERROR in the caller. You cannot specify ERROR within a user-interface trigger block or a destructor. For user-defined functions see note below.
ERROR *return-value*	Undoes the current subtransaction, and raises ERROR in the caller. The CHARACTER string you provide is available to the caller in the RETURN-VALUE function. The AVM also creates an AppError object and stores the *return-value* in the ReturnValue property. For user-defined functions see note below.
ERROR *error-object-expression*	Undoes the current subtransaction, and raises ERROR in the caller. The specified error object instance is thrown to the caller. For user-defined functions see note below.
NO-APPLY	In a user-interface trigger, prevents the AVM from performing the default behavior for the trigger event. Otherwise, the option is ignored.

Note: Using RETURN ERROR in a user-defined function sets the target variable of the function to the Unknown value (?) instead of raising ERROR in the caller. See *ABL Error Handling* for more detail.

Example

In this procedure, if the user presses **END-ERROR** or **END-KEY** while changing the CreditLimit field, any changes made during the current iteration of the block are undone, and the same iteration is run again. If this procedure did not use the ON ENDKEY phrase and the user pressed **END-ERROR**, the procedure ends because the default ENDKEY action is UNDO, LEAVE. After leaving the FOR EACH block, the procedure ends because there are no more statements.

r-endky.p

```
ON WINDOW-CLOSE OF CURRENT-WINDOW
  STOP.

FOR EACH Customer ON ENDKEY UNDO, RETRY:
  DISPLAY Customer.CustNum Customer.Name Customer.CreditLimit.
  SET Customer.CreditLimit
    VALIDATE(Customer.CreditLimit > 0,"non-zero credit limit").
END.
```

See also

ON ERROR phrase, ON QUIT phrase, ON STOP phrase, RETURN statement, RETURN-VALUE function, UNDO statement

ON ERROR phrase

The ON ERROR phrase is one of the ABL constructs used for altering the default condition handling for ABL blocks. By default, when an error occurs in a block, all the changes to persistent data (database fields) are undone. If the block is a transaction block, undo variables and temp-table fields are also undone. See *ABL Essentials* for more information on transaction blocks. The ON ERROR phrase can be used to modify the scope of UNDO processing, possibly widening it to an outer block. In addition, if an error occurs, the ON phrase gives you control over where execution continues.

In general, the AVM performs error handling using this precedence, from highest to lowest. The AVM only abides by one of these when a condition is raised:

- Statement NO-ERROR option
- CATCH block
- Block's ON phrase (explicit or implicit)

Due to this precedence, the ON phrase is ignored if a local CATCH block handles the error.

Syntax

```
ON ERROR UNDO
  [ label1 ]
  [     , LEAVE [ label2 ]
      | , NEXT [ label2 ]
      | , RETRY [ label1 ]
      | , RETURN [ return-value |
                   ERROR [ return-value | error-object-expression ] |
                   NO-APPLY ]
      | , THROW
  ]
```

label1

The name of the block whose processing you want to undo. If you do not name a block with *label1*, the AVM undoes the processing of the current block.

LEAVE [*label2*]

Indicates that after undoing the processing of a block, the AVM leaves the block labeled *label2*. If you do not name a block, the AVM leaves the current block.

NEXT [*label2*]

Indicates that after undoing the processing of a block, the AVM executes the next iteration of the block you name with *label2*. If you do not specify a label with the NEXT option, the AVM executes the next iteration of the current block.

RETRY [*label1*]

Indicates that after undoing the processing of a block, the AVM repeats the same iteration of the block.

Because RETRY in a block without user input results in an infinite loop, the AVM automatically checks for this possibility and converts a RETRY block into a LEAVE block, or a NEXT block if it is an iterating block. This behavior is often referred to as infinite loop protection.

RETURN ...

Returns to the calling routine, if there is one. The following table describes various RETURN options:

Option	Description
return-value	In procedures and VOID methods, this must be a CHARACTER string. The caller can use the RETURN-VALUE function to read the returned value. For user-defined functions, non-VOID methods and property getters, the value must match the specified return type.

ERROR	Undoes the current subtransaction, and raises ERROR in the caller. You cannot specify ERROR within a user-interface trigger block or a destructor. For user-defined functions see note below.
ERROR *return-value*	Undoes the current subtransaction, and raises ERROR in the caller. The CHARACTER string you provide is available to the caller in the RETURN-VALUE function. The AVM also creates an AppError object and stores the *return-value* in the ReturnValue property. For user-defined functions see note below.
ERROR *error-object-expression*	Undoes the current subtransaction, and raises ERROR in the caller. The specified error object instance is thrown to the caller. For user-defined functions see note below.
NO-APPLY	In a user-interface trigger, prevents the AVM from performing the default behavior for the trigger event. Otherwise, the option is ignored.

Note: Using RETURN ERROR in a user-defined function sets the target variable of the function to the Unknown value (?) instead of raising ERROR in the caller. See *ABL Error Handling* for more detail.

THROW

Use this directive to explicitly propagate an error to the enclosing block, if there is one, otherwise to the caller. You can learn more about throwing error objects in *ABL Error Handling*.

Examples

In r-onerr.p, if you enter a Customer number and the FIND statement is unable to find a Customer with that number, the AVM raises an error. If an error occurs, the ON ERROR phrase tells the AVM to undo anything that was done in the current iteration and start the next iteration. Thus, you see any invalid numbers you enter, and you can continue to the next Customer number you want to enter.

r-onerr.p

```
REPEAT ON ERROR UNDO, NEXT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNnum.
  DISPLAY Customer.Name Customer.Address Customer.City Customer.State
    Customer.Country.
END.
```

In r-onErrorThrow01.p the block propagates an error from a DO block up to the main procedure block. A CATCH block on the main procedure block handles the error.

r-onErrorThrow01.p

```
DO ON ERROR UNDO, THROW:
  /* Raises ERROR. The normal error message is diverted to a
     Progress.Lang.SysError error object and thrown to the main block. */
  FIND Customer 1000.
END. /* DO */

MESSAGE "Undisplayed message because of ERROR condition"
  VIEW-AS ALERT-BOX BUTTONS OK.

/* CATCH for main (procedure) block */
CATCH eAnyError AS Progress.Lang.ERROR:
  MESSAGE "Error message and number retrieved from error object..."
    eAnyError:GetMessage(1) eAnyError:GetMessageNum(1)
    VIEW-AS ALERT-BOX BUTTONS OK.
END CATCH.
```

Notes

- For callable blocks, including procedures, user-defined functions, class-based methods, and property accessors use the `ROUTINE-LEVEL ON ERROR UNDO, THROW` or `BLOCK-LEVEL ON ERROR UNDO, THROW` statement, if you want an `UNDO, THROW` directive. `BLOCK-LEVEL ON ERROR UNDO, THROW` is a superset of `ROUTINE-LEVEL ON ERROR UNDO, THROW` and applies to more block types.
- Error objects can be thrown from an application server and handled by a `CATCH` block on an ABL client. To be throwable from an application server to an ABL client, user-defined error classes must be defined on both the server and client sides, and the classes must be defined as `SERIALIZABLE`. For the full list of restrictions on class-based objects that are passed between application server and client, see the Parameter passing syntax entry. For more information on error handling in general, see *ABL Error Handling*.
- If a block does not have implicit error handling capabilities (for example, a simple `DO` or `DO WHILE` block) you can add the `ON ERROR` (or `ON STOP`) phrase to explicitly turn it into one with error handling. This allows you to use a `CATCH` statement within the block to catch any error or stop conditions. If you don't add the `ON ERROR` (or `ON STOP`) phrase and try to use a `CATCH` within the block, you get a compiler error.

See also

BLOCK-LEVEL ON ERROR UNDO, THROW statement,ON ENDKEY phrase, ON QUIT phrase, ON STOP phrase, RETURN statement, RETURN-VALUE function, ROUTINE-LEVEL ON ERROR UNDO, THROW statement, UNDO statement

ON QUIT phrase

Describes the processing that occurs when a `QUIT` statement is executed during a block. By default, the `QUIT` statement saves the current transaction and then returns to the operating system or to the tool from which the routine was invoked (such as the OpenEdge Editor).

Note: Does not apply to SpeedScript programming.

Syntax

```
ON QUIT
  [ UNDO [ label1 ] ]
  [     , LEAVE [ label2 ]
      | , NEXT [ label2 ]
      | , RETRY [ label1 ]
      | , RETURN [ return-value |
                   ERROR [ return-value |error-object-expression] |
                   NO-APPLY ]
  ]
```

`UNDO [ label1 ]`

Indicates that the specified block is undone. If you do not specify the `UNDO` option, the current transaction is committed when the `QUIT` statement executes.

`LEAVE [ label2 ]`

Indicates that after committing or undoing the transaction, the AVM leaves the block labeled *label2*. If you do not name a block, the AVM leaves the current block.

`NEXT [ label2 ]`

Indicates that after committing or undoing the transaction, the AVM executes the next iteration of the block you name with the *label2* option. If you do not name a block with the `NEXT` option, the AVM executes the next iteration of the current block.

`RETRY [ label1 ]`

Indicates that after committing or undoing the processing of a block, the AVM repeats the same iteration of the block that was undone or committed.

Because `RETRY` in a block without user input results in an infinite loop, the AVM automatically checks for this possibility and converts a `RETRY` block into a `LEAVE` block, or a `NEXT` block if it is an iterating block. This behavior is often referred to as infinite loop protection.

`RETURN ...`

Returns to the calling routine, if there is one. The following table describes various `RETURN` options:

Option	Description
return-value	In procedures and VOID methods, this must be a CHARACTER string. The caller can use the `RETURN-VALUE` function to read the returned value. For user-defined functions, non-VOID methods and property getters, the value must match the specified return type.

ERROR	Undoes the current subtransaction, and raises ERROR in the caller. You cannot specify ERROR within a user-interface trigger block or a destructor. For user-defined functions see note below.
ERROR *return-value*	Undoes the current subtransaction, and raises ERROR in the caller. The CHARACTER string you provide is available to the caller in the RETURN-VALUE function. The AVM also creates an AppError object and stores the *return-value* in the ReturnValue property. For user-defined functions see note below.
ERROR *error-object-expression*	Undoes the current subtransaction, and raises ERROR in the caller. The specified error object instance is thrown to the caller. For user-defined functions see note below.
NO-APPLY	In a user-interface trigger, prevents the AVM from performing the default behavior for the trigger event. Otherwise, the option is ignored.

Note: Using RETURN ERROR in a user-defined function sets the target variable of the function to the Unknown value (?) instead of raising ERROR in the caller. See *ABL Error Handling* for more detail.

See also

ON ENDKEY phrase, ON ERROR phrase, ON STOP phrase, QUIT statement, RETURN statement, RETURN-VALUE function, UNDO statement

ON statement

The ON statement specifies a trigger for one or more events or redefines terminal keys for an application.

Syntax

```
ON event-list
  {     ANYWHERE
     |  { OF widget-list
             [ OR event-list OF widget-list ] . . .
             [ ANYWHERE ]
        }
  }
  {     trigger-block
     |  REVERT
     |  { PERSISTENT RUN procedure
             [ ( input-parameters ) ]
        }
  }
```

```
ON event OF database-object
  [referencing-phrase]
  [ OVERRIDE ]
  {trigger-block | REVERT }
```

```
ON key-labelkey-function
```

```
ON "WEB-NOTIFY" ANYWHERE  {trigger-block}
```

event-list

A comma-separated list of user-interface or developer events for which you want to define a trigger. If any of the specified events occurs for any of the specified widgets, the trigger executes.

For a list of valid events for each widget type, see the reference page for that widget type. For information on all user interface events, see the Handle-based Object Events Reference.

In batch mode (running with the -b startup parameter) and when running on an application server, the ON statement supports only the following events:

- PROCEDURE-COMPLETE
- READ-RESPONSE
- CONNECT
- Developer events (U1 to U10, and CLOSE)

  The ten miscellaneous developer events (U1 to U10) have no standard meaning or action. They are reserved for an ABL developer to use for associating a trigger with some event that only happens programmatically using the APPLY statement. They have nothing to do with a particular user action and have no built-in significance. The CLOSE event has these same charactertistics and only occurs when specified with the APPLY statement.

  OpenEdge clients running in batch mode process these events just like non-batch clients do.

widget-list

A comma-separated list of widgets or procedure handles to which the event is applied. See the Widget phrase reference entry for more information on referencing widgets.

If a specified event occurs for any of the specified widgets, the trigger executes. If you specify a list of widgets, all events specified must be user-interface events.

ANYWHERE

You can specify ANYWHERE either with a list of widgets or instead of a list of widgets. Without a list of widgets, ANYWHERE specifies that the trigger executes when one of the specified events occurs for any widget that does not already have a specific trigger for that event. This lets you define a default trigger for the event within the application. With a list of widgets, ANYWHERE specifies that the trigger executes when one of the specified events occurs for any specified widget or for any contained widget that does not already have a specific trigger for that event. This lets you set up a default trigger for a frame or window.

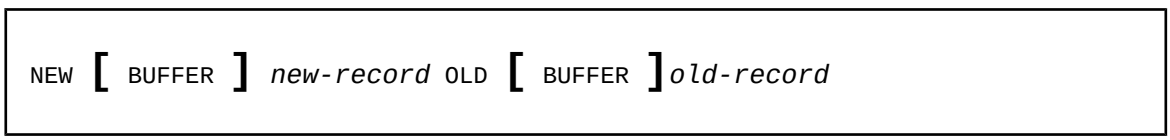
event

A database event: CREATE, DELETE, FIND, WRITE or ASSIGN. If the specified event occurs for the specified table or field, the trigger executes. For database events, you can specify only one event.

database-object [*referencing-phrase*]

The name of a database table or field to which the event is applied. If you specify a *database-object*, the event specified must be a database event. You cannot specify a metaschema table or field (a table or field named with an initial underscore) as the *database-object.*

The *referencing-phrase* is valid only for WRITE and ASSIGN triggers. For WRITE triggers you can specify a name for the record before the WRITE operation and a name for the record after the WRITE operation. This allows you to reference both versions of the record within the trigger. This is the syntax for WRITE trigger:



For an ASSIGN trigger, you can specify a name for the old field value. This is the syntax:

```
OLD [ VALUE ] old-field-name
```

OVERRIDE

Specifies that the database trigger you are defining overrides the schema trigger for the same event. You can override a schema trigger only if it is defined as overridable in the Data Dictionary. If you do not use the OVERRIDE option, then the session trigger executes first and then the schema trigger.

trigger-block

A trigger block is either a single ABL statement or a set of statements grouped by DO and END statements. The trigger block is executed when one of the specified events is applied to one of the specified widgets or tables.

REVERT

If you specify this option, any non-persistent trigger defined in this procedure for the event is reverted. If a trigger had also been defined for the event in a previous procedure, that previous trigger again takes effect. The AVM ignores any attempt to revert a persistent trigger.

PERSISTENT RUN *procedure* [(*input-parameters*)]

Specifies a persistent trigger; that is, a trigger that remains in effect after the current procedure terminates. Normally, a trigger remains in effect only until the procedure or trigger in which it is defined ends. You can specify a persistent trigger only for user-interface events. A persistent trigger must be a procedure specified by *procedure*. The trigger procedure can take one or more input parameters; it cannot have any output parameters. The parameters of the trigger procedure are evaluated when you define the trigger; they are **not** re-evaluated when the trigger executes.

key-label

The label of the key for which you want to define a specific action. See *OpenEdge Programming Interfaces* for a list of key labels.

On UNIX, all of the special ABL keys are defined in the PROTERMCAP file supplied with ABL. If the key for which you are defining an action is not already in PROTERMCAP, you must add a definition for that key. Keys that you can name that do not require a PROTERMCAP definition are **CTRL**, **RETURN**, **BACKSPACE**, **TAB**, and **DEL**.

In Windows, keys are predefined as described in the handling user input section of *OpenEdge Programming Interfaces*.

key-function

The action you want the AVM to take when the user presses the key associated with *key-label.* The *key-function* value can be one of the key functions listed in the following table.

Table 48: Valid key functions

ABORT	BACKSPACE	BACK-TAB
BELL	CLEAR	CURSOR-DOWN

CURSOR-LEFT	CURSOR-RIGHT	CURSOR-UP
DELETE-CHARACTER	END	END-ERROR
ENDKEY	ENTER-MENUBAR	ERROR
GO	HELP	HOME
INSERT-MODE	LEFT-END	NEXT-FRAME
PREV-FRAME	RECALL	RETURN
RIGHT-END	SCROLL-MODE	STOP
TAB	–	–

Examples

The following example defines a WRITE trigger for the customer table:

r-oncst.p

```
ON WRITE OF Customer NEW new-cust OLD old-cust DO:
  IF new-cust.City <> old-cust.City AND
    new-cust.PostalCode = old-cust.PostalCode THEN DO:
    MESSAGE "Must update postal code, too.".
    RETURN ERROR.
  END.
END.

FOR EACH Customer:
  UPDATE Customer.
END.
```

The trigger compares the Customer record before the write with the Customer record after the write. If the city has changed and the postal code has not changed, the trigger displays a message and cancels the write operation.

The following example uses the ON statement to set up a trigger for two buttons:

r-widget.p

```
DEFINE BUTTON b_next LABEL "Next".
DEFINE BUTTON b_prev LABEL "Previous".
DEFINE BUTTON b_quit LABEL "Quit".

DEFINE FRAME butt-frame
  b_next b_prev
  WITH CENTERED ROW SCREEN-LINES - 1.
DEFINE FRAME info
  Customer.CustNum Customer.Name
  b_quit AT ROW-OF Customer.CustNum + 2 COLUMN-OF Customer.CustNum + 18
  WITH CENTERED TITLE "Customers"  ROW 2 1 COL.

ON CHOOSE OF b_next, b_prev DO:
  IF SELF:LABEL = "Next" THEN
    FIND NEXT Customer NO-LOCK.
  ELSE
    FIND PREV Customer NO-LOCK.
  DISPLAY Customer.CustNum Customer.Name WITH FRAME info.
END.

ENABLE b_next b_prev WITH FRAME butt-frame.
ENABLE b_quit WITH FRAME info.

WAIT-FOR END-ERROR OF FRAME butt-frame OR
  CHOOSE OF b_quit IN FRAME info FOCUS b_next IN FRAME butt-frame.
```

The following procedure sets up mappings for **GO**, **HELP**, and **END** and defines **CTRL+X** to ring the terminal bell:

r-onstmt.p

```
ON F1 GO.       /* F1 will now perform the GO function */
ON F2 HELP.     /* F2 will now perform the HELP function   */
ON CTRL-X BELL. /* The Ctrl-X key will be disabled */
ON F5 ENDKEY.   /* F5 will always raise the ENDKEY condition; never ERROR*/
```

Notes

- If you use the ON statement to redefine terminal keys, the new definitions remain in effect to the end of the session or until another ON statement changes the definition.
- A trigger defined with the ON statement remains in effect until one of the following occurs:
  - Another ON statement defines another trigger (or REVERT) for the same event and widget
  - For a non-persistent trigger, the procedure or trigger block in which the ON statement appears terminates
- Although each widget type responds with default system actions to a limited set of valid events, you can specify any event for any widget and execute the trigger using the APPLY statement. If the event is not a valid event for the widget type, the specified trigger executes, but no default system action occurs for the widget. You can use this feature to write triggers for procedure handles that do not otherwise respond to events.
- If *event-list* includes a MENU-DROP event for a menu or submenu, do not interact with the window manager from within the *trigger-block*. Doing so causes the window manager to lose control of the system, forcing you to reboot or restart the window manager. Actions to avoid include any window system input/output (I/O) or any lengthy processing, especially in statements that cause process interruptions, such as the PAUSE statement with or without I/O. These also include actions that can generate a warning or error message, forcing window system output. Use the NO-ERROR option on supported statements to help

avoid this situation. Otherwise, check valid values, especially for run-time resources like handles, to prevent the AVM from displaying unexpected messages.

- For SpeedScript, the only valid uses of the ON statement are specifying a trigger for a database event or for specifying a trigger for a WEB-NOTIFY event (the ON "WEB-NOTIFY" ANYWHERE syntax).
- The ON statement only works with ABL events. You cannot use the ON statement to interact with .NET object events.

See also

APPLY statement, Developer events, Widget phrase

ON STOP phrase

Describes processing that occurs when the STOP condition is raised. The STOP condition is raised when:

- The user presses the **STOP** key (usually mapped to **CTRL+BREAK** in Windows and **CTRL+C** on Unix/Linux.)
- The time-out is exceeded for a STOP-AFTER directive on a DO, FOR, or REPEAT statement.
- The time-out set using the Lock Timeout (-lkwtmo) startup parameter is exceeded when waiting for release of a record lock, or when the user indicates they want to stop waiting.
- A STOP statement executes.
- System errors occur that raise the STOP (instead of the ERROR) condition.

By default, when a STOP condition occurs in a block, all the changes to persistent data (database fields) are undone. If the block is a transaction block, undo variables and temp-table fields are also undone. See *ABL Essentials* for more information on transaction blocks. The ON STOP phrase can be used to modify the scope of UNDO processing, possibly widening it to an outer block.

Also by default, a STOP condition causes execution to leave the current block. The STOP condition is continually propagated to the outer block, or up the call stack if there is no outer block. The ON STOP phrase can be used to control where execution continues and to clear the STOP condition so that it is not propagated beyond the current block.

The AVM performs stop handling using this precedence, from highest to lowest. The AVM only abides by one of these when a STOP is raised:

- CATCH block
- Block's ON phrase (explicit or implicit)

Due to this precedence, the ON STOP phrase is ignored if a local CATCH block handles the stop.

Syntax

```
ON STOP UNDO
  [ label1 ]
  [      , LEAVE [ label2 ]
       | , NEXT [ label2 ]
       | , RETRY [ label1 ]
       | , RETURN [ return-value |
                    ERROR [ return-value | error-object-expression ] |
                    NO-APPLY ]
  ]
```

label1

The name of the block whose processing you want to undo. If you do not name a block with *label1*, the AVM undoes the processing of the current block.

LEAVE [*label2*]

Indicates that, after undoing the processing of a block, the AVM leaves the block labeled *label2*. If you do not name a block, the AVM leaves the current block. There are restrictions. For example, you cannot undo an outer block, but leave only the inner block.

NEXT [*label2*]

Indicates that, after undoing the processing of a block, the AVM executes the next iteration of the block you name with the *label2* option. If you do not name a block with the NEXT option, the AVM executes the next iteration of the current block.

RETRY [*label1*]

Indicates that after undoing the processing of a block, the AVM repeats the same iteration of the block.

Because RETRY in a block without user input results in an infinite loop, the AVM automatically checks for this possibility and converts a RETRY block into a LEAVE block, or a NEXT block if it is an iterating block. This behavior is often referred to as infinite loop protection.

RETURN ...

Returns to the calling routine, if there is one. The following table describes various RETURN options:

Option	Description
return-value	In procedures and VOID methods, this must be a CHARACTER string. The caller can use the RETURN-VALUE function to read the returned value. For user-defined functions, non-VOID methods and property getters, the value must match the specified return type.

ERROR	Undoes the current subtransaction, and raises ERROR in the caller. You cannot specify ERROR within a user-interface trigger block or a destructor. For user-defined functions see note below.
ERROR *return-value*	Undoes the current subtransaction, and raises ERROR in the caller. The CHARACTER string you provide is available to the caller in the RETURN-VALUE function. The AVM also creates an AppError object and stores the *return-value* in the ReturnValue property. For user-defined functions see note below.
ERROR *error-object-expression*	Undoes the current subtransaction, and raises ERROR in the caller. The specified error object instance is thrown to the caller. For user-defined functions see note below.
NO-APPLY	In a user-interface trigger, prevents the AVM from performing the default behavior for the trigger event. Otherwise, the option is ignored.

Note: Using RETURN ERROR in a user-defined function sets the target variable of the function to the Unknown value (?) instead of raising ERROR in the caller. See *ABL Error Handling* for more detail.

Example

This procedure lets you update the CreditLimit field for each Customer. If you enter a value greater than 100,000, the program raises the STOP condition. Since you specified an UNDO, RETRY for a STOP, the procedure starts the iteration over and allows you to enter another value.

r-ostop.p

```
FOR EACH Customer ON STOP UNDO, RETRY:
  DISPLAY Customer.CustNum Customer.Name Customer.CreditLimit.
  UPDATE Customer.CreditLimit.
  IF Customer.CreditLimit > 100000 THEN STOP.
END.
```

Notes

- When both a local ON STOP phrase and a matching local CATCH block is specified on the current block, the local CATCH block takes precedence and the ON STOP phrase is ignored. If no local CATCH block matches the thrown stop object type, the specified directive in the local ON STOP phrase executes instead for any raised STOP condition.
- Almost all STOP conditions are trappable with the ON STOP phrase or a CATCH block. In some cases, the AVM might ignore STOP conditions at certain levels of the call stack. For example, if the AVM executes a procedure that relies on a database and the connection is lost, the AVM raises the STOP condition and unwinds the call stack until it gets to a level **above** all references to the database. ON STOP phrases and compatible CATCH blocks before this point are ignored. After this point, the AVM executes the directive

specified in the ON STOP phrase or compatible CATCH block, with the CATCH block taking precedence if the ON STOP phrase is on the same associated block.

- If a block does not have implicit error handling capabilities (for example, a simple DO or DO WHILE block) you can add the ON STOP (or ON ERROR) phrase to explicitly turn it into one with error handling. This allows you to use a CATCH statement within the block to catch any error or stop conditions. If you don't add the ON STOP (or ON ERROR) phrase and try to use a CATCH within the block, you get a compiler error.

See also

CATCH statement, ON ENDKEY phrase, ON ERROR phrase, ON QUIT phrase, Progress.Lang.Stop class, Progress.Lang.StopError class, RETURN statement, RETURN-VALUE function, STOP statement

OPEN QUERY statement

Opens a query, which might have been previously defined in a DEFINE QUERY statement. Opening a query makes it available for use within a GET statement, or in a browse widget.

Syntax

```
OPEN QUERY query { FOR | PRESELECT } EACH record-phrase
  [ , { EACH | FIRST | LAST } record-phrase ] ...
  [ query-tuning-phrase ]
  [ BREAK ]
  [ BY expression [ DESCENDING ]
    | COLLATE ( string , strength [ , collation ] ) [ DESCENDING ]
  ] ...
  [ INDEXED-REPOSITION ]
  [ MAX-ROWS num-results ]
```

query

The query to open. The query name may have been defined previously in a DEFINE QUERY statement. Otherwise, the OPEN QUERY statement implicitly defines the query.

{ FOR **|** PRESELECT **}** EACH *record-phrase*

Specifies the first buffer of the query.

The following is the syntax for *record-phrase*:

```
record
  [ [ LEFT ] ] [ OF table ]
  [ WHERE expression ]
  [ TENANT-WHERE expression [ SKIP-GROUP-DUPLICATES ]]
  [ USING [ FRAME frame ] field
      [ AND [ FRAME frame ] field ] ... ]
  [ USE-INDEX index ]
  [ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ]
  [ NO-PREFETCH ]
```

If the query was previously defined, the buffers referenced by the *record-phrase* must be the same buffers referenced in the DEFINE QUERY statement and in the same order. For more information, see the Record phrase reference entry.

Note that the first buffer must be qualified with EACH rather than the FIRST option. That is, the OPEN QUERY statement implies the possibility of a multi-row result, whether or not only one row is returned.

If you specify PRESELECT rather than FOR, then the AVM preselects the records for the query. During the preselect process, the AVM applies whatever locking is specified in the OPEN QUERY statement or, if none is specified, SHARE-LOCK. It then reads the ROWID for each record into the result list. (If you do not specify PRESELECT, the AVM might pass through the records anyway to presort them. In this case, the AVM applies NO-LOCK to each record during this pass.)

```
{ EACH | FIRST | LAST }record-phrase
```

Specifies subsequent buffers in the query. Each subsequent buffer specifies a join with the previous buffer(s) according to the *record-phrase*. If the query was previously defined, the buffers referenced by the *record-phrase* must be the same buffers referenced in the DEFINE QUERY statement and in the same order. For more information on specifying joins in Record phrases, see the Record phrase reference entry.

query-tuning-phrase

Allows programmatic control over the execution of a DataServer query. Following is the syntax for the *query-tuning-phrase*:

```
QUERY-TUNING
  (
    [ LOOKAHEAD [ CACHE-SIZE integer ] | NO-LOOKAHEAD ]
    [ DEBUG { SQL | EXTENDED } | NO-DEBUG ]
    [ SEPARATE-CONNECTION | NO-SEPARATE-CONNECTION ]
    [ JOIN-BY-SQLDB | NO-JOIN-BY-SQLDB ]
    [ BIND-WHERE | NO-BIND-WHERE ]
    [ INDEX-HINT | NO-INDEX-HINT ]
  )
```

For more information, see your OpenEdge DataServer Guides (*Use the Microsoft SQL Data Server* and *Use the Oracle Data Server*).

BREAK

Over a series of query iterations, you might want to do some work based on whether the value of a certain field changes. This field defines a break group. For example, you might be accumulating some value, such as a total. You use the BREAK option to define customer.state as the break group, as shown:

```
OPEN QUERY q-order FOR EACH customer BREAK BY customer.state NO-LOCK.
```

When using the BREAK option you must also use the BY option to name a sort field.

To test whether a break group has changed, you can use the FIRST-OF( ) method and LAST-OF( ) method of the query object handle.

BY *expression* [DESCENDING]

Specifies the order in which records are to be returned. If an index is defined with the right leading keys to satisfy the BY clause, the AVM uses that index to sort the records. Otherwise, the AVM must presort the records before the first fetch when you specify BY. The DESCENDING option sorts the records in descending order (not in the default ascending order).

COLLATE (*string* , *strength* [, *collation*]) [DESCENDING]

Generates the collation value of a string after applying a particular strength, and optionally, a particular collation. The DESCENDING option sorts the records in descending order (not in default ascending order).

string

A CHARACTER expression that evaluates to the string whose collation value you want to generate.

strength

A CHARACTER expression that evaluates to an ABL comparison strength or an International Components for Unicode (ICU) comparison strength.

The ABL comparison strengths include:

- **RAW** — Generates a collation value for the string based on its binary value.
- **CASE-SENSITIVE** — Generates a case-sensitive collation value for the string based on a particular collation. If you specify this strength with an ICU collation, the AVM applies the ICU TERTIARY strength.
- **CASE-INSENSITIVE** — Generates a case-insensitive collation value for the string based on a particular collation. If you specify this strength with an ICU collation, the AVM applies the ICU SECONDARY strength.
- **CAPS** — Generates a collation value for the string based on its binary value after converting any lowercase letters in the string to uppercase letters, based on the settings of the Internal Code Page (`-cpinternal`) and Case Table (`-cpcase`) startup parameters.

The ICU comparison strengths include:

- **PRIMARY** — Generates a collation value for the base characters in the string.
- **SECONDARY** — Generates a collation value for the base characters and any diacritical marks in the string.
- **TERTIARY** — Generates a case-sensitive collation value for the base characters and any diacritical marks in the string.
- **QUATERNARY** — Generates a case-sensitive collation value for the base characters and any diacritical marks in the string, and distinguishes words with and without punctuation. ICU uses this strength to distinguish between Hiragana and Katakana when applied with the ICU-JA (Japanese) collation. Otherwise, it is the same as TERTIARY.

Note: Use ICU comparison strengths only with ICU collations.

collation

A CHARACTER expression that evaluates to the name of an ABL collation table or ICU collation. If *collation* does not appear, COLLATE uses the collation table of the client.

- The AVM reports an error and stops execution if one of the following occurs:
  - *strength* does not evaluate to a valid value.
  - *collation* does not evaluate to a collation table residing in the `convmap.cp` file.
  - *collation* evaluates to a collation table that is not defined for the code page corresponding to the `-cpinternal` startup parameter.

`INDEXED-REPOSITION`

If you specify this option, the AVM attempts to optimize subsequent REPOSITION TO ROWID operations on the query. This can improve the performance of REPOSITION operations that must jump over many records in a simple query. Optimization is not possible if the database is not an OpenEdge database, or sorting or preselection is performed. In these cases, the INDEXED-REPOSITION option is ignored and no error is reported.

The optimization has some side effects. When you perform a REPOSITION TO ROWID with this optimization, the AVM discards the original result list and begins a new one. Therefore, scrolling forward or backward in the list might return different records from before. Also, the values of the NUM-RESULTS and CURRENT-RESULT-ROW become invalid. If the query has an associated browse, any selections in that browse are also lost. Lastly, the vertical scrollbar thumb is disabled. Because of these side-effects, use this option selectively.

`MAX-ROWS` *num-results*

Specifies the maximum number of records to be returned by the query. Any other records satisfying the query are ignored and no error is raised. The limit is imposed before any sorting occurs; the AVM retrieves records up to the number specified and then sorts those records.

This option is valid for scrolling queries only. You can use it to prevent a long delay that might occur if a query returns many more records than you expect.

Example

The following example opens a query on the Customer, Order, OrderLine, and Item tables:

r-opqury.p

```
DEFINE QUERY q-order FOR Customer, Order, OrderLine, Item.

OPEN QUERY q-order FOR EACH Customer,
  EACH Order OF Customer,
  EACH OrderLine OF Order,
  EACH Item OF OrderLine NO-LOCK.

GET FIRST q-order.

DO WHILE AVAILABLE Customer:
  DISPLAY Customer.CustNum Customer.Name SKIP
    Customer.Phone SKIP
    Order.OrderNum Order.OrderDate SKIP
    OrderLine.LineNum OrderLine.Price OrderLine.Qty SKIP
    Item.ItemNum Item.ItemName SKIP
    Item.CatDesc VIEW-AS EDITOR SIZE 50 BY 2 SCROLLBAR-VERTICAL
    WITH FRAME ord-info CENTERED SIDE-LABELS TITLE "Order Information".

  /* Allow scrolling, but not modification, of CatDesc. */
  ASSIGN Item.CatDesc:READ-ONLY IN FRAME ord-info = TRUE
         Item.CatDesc:SENSITIVE IN FRAME ord-info = TRUE.

  PAUSE.
  GET NEXT q-order.
END. /* DO WHILE AVAILABLE Customer */
```

Note the use of field lists in the DEFINE QUERY statement. This can improve the performance of remote database queries significantly.

Notes

- If the query you reference in an OPEN QUERY statement is already open, then that query is closed and a new query is opened.
- If you use the USE-INDEX option of the Record phrase, the AVM uses only that index. Records are returned in index order.
- The locking options of the OPEN QUERY statement define the default locking for records fetched by the query. You can override the default by using a locking option in the GET statement. Note, however, that in the OPEN QUERY statement you can specify a separate lock type for each buffer; in the GET statement you can specify only one lock type that applies to all buffers in a join.
- The record locking behavior specified for a query in the DEFINE BROWSE statement overrides the record locking behavior specified with the OPEN QUERY statement. The default record locking behavior of a browse widget is NO-LOCK. The default record locking behavior of a query defined with the OPEN QUERY statement is SHARE-LOCK. If you define a query and a browse widget for the query without explicitly defining record locking behavior, the query will have the NO-LOCK behavior.
- Each time you open a query associated with a browse widget, the data in the browse is refreshed.
- If you reopen a query to refresh the data displayed in a browse widget, a selected row might not reflect the database's most current data. To view the latest data in a selected row, add a FIND CURRENT statement before the OPEN QUERY statement. This additional code allows the AVM to put the current record into memory before executing the OPEN QUERY statement.
- You cannot use the CAN-FIND function in a WHERE clause. Doing so generates a compiler error.
- If you open a query that has already been defined with multiple buffers, you must specify the buffers in the same order in the OPEN QUERY as they were specified in the DEFINE QUERY statement.

- Once the query has been opened, you cannot change the buffers that it references, even if the query is closed and re-opened. For example, a buffer, buff1, is created for the customer table in a DEFINE QUERY or OPEN QUERY for the query, qry1. The query is run and closed. You cannot now DEFINE or OPEN qry1 with buff1 for the item table. You can reuse buffers with CREATE QUERY, but you must re-run QUERY-PREPARE.
- A ProDataSet data-relation defined with REPOSITION or SELECTION is overridden if a query is defined for a child relation data-source. Normally, if REPOSITION is not specified, the child query selects the children of the parent. But coding a query for the child data-source overrides whether or not a SELECTION or REPOSITION relation mode was defined.

See also

CLOSE QUERY statement, CREATE QUERY statement, CURRENT-RESULT-ROW function, DEFINE BROWSE statement, DEFINE DATASET statement, DEFINE QUERY statement, GET statement, NUM-RESULTS function, QUERY-OFF-END function, QUERY-PREPARE( ) method, REPOSITION statement

OPSYS function

Identifies the operating system being used, so that a single version of a procedure can work differently under different operating systems. Returns the value of that operating system. Valid values are "UNIX" and "WIN32".

Note: For both 32-bit and 64-bit Windows operating systems, OPSYS returns the value "WIN32".

Syntax

```
OPSYS
```

Example

This procedure produces a listing of the files in your current directory. The OPSYS function determines which operating system you are running, and uses the appropriate operating system command to produce the directory listing. The example shows the possible return values.

r-opsys.p

```
IF OPSYS = "UNIX" THEN UNIX ls.
ELSE IF OPSYS = "WIN32" THEN DOS dir.
ELSE MESSAGE OPSYS "is an unsupported operating system".
```

Note

ABL supports an override option that enables applications that need to return the value of MS-DOS for all Microsoft operating systems to do so. For example, if you do not want the value WIN32 returned when the Windows operating system is recognized, you can override this return value by defining the Opsys key in the Startup section of the current environment, which may be in the registry or in an initialization file. If the Opsys key is located, the OPSYS function returns the value associated with the Opsys key on all platforms.

See also

DOS statement, UNIX statement, PROCESS-ARCHITECTURE function, { } Preprocessor name reference

OR operator (bitwise)

Performs a bitwise inclusive OR operation on two expressions of the same flag enum type and returns a new instance of the same type.

Syntax

```
flag-enum-expression OR flag-enum-expression
```

flag-enum-expression

An expression whose value is a flag enum instance. Both expressions must be the same flag enum type, and neither expression can be the Unknown value (?).

Example

You can use bitwise OR to set multiple flags. This code fragment defines a variable of type `Progres.Reflect.Flags` (see the Progress.Reflect.Flags enumeration entry for a full list of enumerators) and uses OR to set both the `Public` and `Protected` flags.

```
DEFINE VARIABLE flagsEnum AS Progress.Reflect.Flags.

flagsEnum = Flags:Public OR Flags:Protected.
```

The following example shows bitwise OR being used with one explicitly named flag and a variable defined as type `Progress.Reflect.Flags`. The first bitwise OR sets the `Static` and `Public` flags of `flagsEnum1`. The second bitwise OR results in `flagsEnum2` having the `Static`, `Public`, and `Instance` flags set.

```
DEFINE VARIABLE flagsEnum1 AS Progress.Reflect.Flags.
DEFINE VARIABLE flagsEnum2 AS Progress.Reflect.Flags.

flagsEnum1 = Flags:Static OR Flags:Public.
flagsEnum2 = Flags:Instance OR flagsEnum1.
```

Notes

- This operator can be used with both ABL and .NET flag enums.

See also

AND operator (bitwise), ENUM statement, Expression, NOT operator (bitwise), XOR operator (bitwise)

OR operator (logical)

Returns a TRUE value if either of two logical expressions is TRUE.

Syntax

```
expression OR expression
```

expression

A logical expression (a constant, field name, variable name or expression whose value is logical, that is, TRUE/FALSE, YES/NO).

Example

This procedure lists Customers who have no postal code (`PostalCode = ""`) or that have no telephone number (`Phone = ""`), and displays how many Customers are in the list:

r-or.p

```
FOR EACH Customer NO-LOCK
  WHERE Customer.PostalCode = "" OR Customer.Phone = "":
  DISPLAY Customer.CustNum Customer.Name (COUNT) Customer.City Customer.State
    Customer.PostalCode Customer.Phone.
END.
```

See also

AND operator (logical), NOT operator (logical)

OS-APPEND statement

Executes an operating system file append command from within ABL.

Syntax

```
OS-APPEND
  { source-filename | VALUE ( expression ) }
  { target-filename | VALUE ( expression ) }
```

source-filename

The name of the source file. (If you append file A to file B, file A is the source file.) If you specify a directory, OS-APPEND generates an error. The filename can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

VALUE (*expression*)

An expression that returns the name of the source file. (If you append file A to file B, file A is the source file.) *expression* can contain constants, field names, and variable names. The filename can contain Unicode characters.

target-filename

The name of the target file. (If you append file A to file B, file B is the target file.) The filename can contain Unicode characters.

VALUE (*expression*)

An expression that returns the name of the target file. (If you append file A to file B, file B is the target file.) *expression* can contain constants, field names, and variable names. The filename can contain Unicode characters.

Example

This procedure opens a dialog box that prompts the user to choose a source file for the append. It then prompts for a name for the target file. Finally, the procedure uses the OS-APPEND statement to append the source file to the target file.

r-os-app.p

```
DEFINE VARIABLE sourcefile AS CHARACTER NO-UNDO.
DEFINE VARIABLE targetfile AS CHARACTER NO-UNDO FORMAT "x(20)"
  VIEW-AS FILL-IN.
DEFINE VARIABLE OKpressed  AS LOGICAL   NO-UNDO INITIAL TRUE.

Main:
REPEAT:
  SYSTEM-DIALOG GET-FILE sourcefile
    TITLE "Choose Source File For Append"
    MUST-EXIST
    USE-FILENAME
    UPDATE OKpressed.

  IF OKpressed = FALSE THEN
    LEAVE Main.
  UPDATE targetfile WITH FRAME appendframe.
  OS-APPEND VALUE(sourcefile) VALUE(targetfile).
END.
```

Notes

- The filenames must conform to the naming conventions of the underlying operating system.
- If *target-file* names a file that does not exist or a directory, OS-APPEND becomes an OS-COPY and a copy is created in the current or specified directory. If an error occurs during the copy, the AVM deletes the partial *target-file*.
- Although an error can occur during execution of this statement, the statement does not generate an error message, raise an error condition, or affect the program's flow in any way. Check for an execution error by using the OS-ERROR function and evaluating the return.
- If you specify the same file for the source and the target, the append fails but OS-ERROR is not set.

See also

OS-ERROR function

OS-COMMAND statement

Escapes to the current operating system and executes an operating system command.

Syntax

```
OS-COMMAND
   [ SILENT | NO-WAIT | NO-CONSOLE ]
   [ command-token | VALUE ( expression ) ]...
```

SILENT

After processing an operating system command, the AVM shell pauses. To exit the window in Windows GUI platforms, you must type `exit`. To exit the window in Windows character platforms, you must type `exit` and press **RETURN** or **SPACEBAR**. You can use the SILENT option to eliminate this pause. Use this option only if you are sure that the program, command, or batch file does not generate any output to the screen. Cannot be used with NO-WAIT.

NO-WAIT

In a multi-tasking environment, causes the AVM to immediately pass control back to next statement after the OS-COMMAND without waiting for the operating system command to terminate. Cannot be used with SILENT. This option is supported in Windows only.

NO-CONSOLE

While processing an operating system command, the AVM creates a console window. The console window may not be cleaned up after the command is executed. You can use the NO-CONSOLE option to prevent this window from being created in the first place.

command-token | VALUE (*expression*)

One or more command words and symbols that you want to pass the operating system to execute. The VALUE option generates the command tokens included in *expression*, a character string expression. The specified combination of *command-token* and VALUE(*expression*) options can form any legal combination of commands and command options permitted by the operating system. These can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

Example

There are two principal uses for the OS-COMMAND statement: to execute a utility that has the same syntax on two or more different operating systems, and to execute an operating system statement input by a user.

In both instances, the OS-COMMAND statement eliminates the need to use the OPSYS statement to determine the operating system and then use conditional logic to execute the appropriate code. The OS-COMMAND statement, therefore, makes an application more portable.

This procedure prompts the user for an operating system command and then uses the OS-COMMAND statement to execute the command:

r-os-com.p

```
DEFINE VARIABLE comm-line AS CHARACTER NO-UNDO FORMAT "x(70)".

REPEAT:
  UPDATE comm-line.
  OS-COMMAND VALUE(comm-line).
END.
```

Notes

- If you want to run an operating system internal command, such as Windows `dir`, do not use the NO-WAIT keyword. The results are unpredictable.
- If you want to run an application that requires Windows, you must use the NO-WAIT option.
- The NO-WAIT option is unavailable in environments that are not multi-tasking.
- The OS-COMMAND statement always sets the value for the OS-ERROR function to 0, whether or not an error occurs. Thus, an operating system error is never returned for the OS-COMMAND statement.

See also

DOS statement, OPSYS function, OS-ERROR function, UNIX statement

OS-COPY statement

Executes an operating system file copy command from within ABL.

Syntax

```
OS-COPY
  { source-filename | VALUE ( expression ) }
  { target-filename | VALUE ( expression ) }
```

source-filename

The name of the original file. If you specify a directory, OS-COPY generates an error. The filename can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

VALUE (*expression*)

An expression that returns the name of the original file. *Expression* can contain constants, field names, and variable names. The filename can contain Unicode characters.

target-filename

The name of the new file or directory. If you specify a directory, OS-COPY gives the target file the same name as the source file. The filename can contain Unicode characters.

VALUE (*expression*)

An expression that returns the name of the new file or directory. *expression* can contain constants, field names, and variable names. The filename can contain Unicode characters.

Example

This procedure opens a dialog box that prompts the user to choose a file to copy. It then prompts for a name for the copy. Finally, the procedure uses the OS-COPY statement to copy the file.

r-os-cop.p

```
DEFINE VARIABLE sourcefilename AS CHARACTER NO-UNDO.
DEFINE VARIABLE copyfilename   AS CHARACTER NO-UNDO FORMAT "x(20)"
  VIEW-AS FILL-IN.
DEFINE VARIABLE OKpressed      AS LOGICAL   NO-UNDO INITIAL TRUE.

Main:
REPEAT:
  SYSTEM-DIALOG GET-FILE sourcefilename
    TITLE  "Choose File to Copy"
    MUST-EXIST
    USE-FILENAME
    UPDATE OKpressed.

  IF OKpressed = FALSE THEN
    LEAVE Main.
  UPDATE copyfilename WITH FRAME copyframe.
  OS-COPY VALUE(sourcefilename) VALUE(copyfilename).
END.
```

Notes

- The filenames must conform to the naming conventions of the underlying operating system.
- If *target-file* specifies an existing file, OS-COPY overwrites the existing file.
- If *target-file* has the same name as *source-file*, the copy fails, but OS-ERROR is not set.
- If the copy terminates abnormally, the AVM deletes the partial *target-file*.
- Enclose filenames that refer to physical devices in double quotes (" ").
- Although an error can occur during execution of this statement, the statement does not generate an error message, raise an error condition, or affect the program's flow in any way. Check for an execution error by using the OS-ERROR function and evaluating the return.

See also

OS-ERROR function

OS-CREATE-DIR statement

Executes an operating system command from within ABL that creates one or more new directories.

Syntax

```
OS-CREATE-DIR { dirname | VALUE ( expression ) }...
```

dirname

The name of a directory to create. If the directory already exists, no error is generated. If a file with this name exists, an error is generated. The name can be a pathname or a simple name. The directory name can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

If *dirname* is not a fully qualified directory name, ABL prepends the current working directory to *dirname*.

VALUE (*expression*)

An expression that returns the name of a directory to create. *Expression* can contain constants, field names, and variable names. The directory name can contain Unicode characters.

Example

The following procedure prompts the user for the name of a directory, then creates it. If the name you give is not fully qualified, the directory is created in your current directory.

r-os-dir.p

```
DEFINE VARIABLE stat     AS INTEGER   NO-UNDO.
DEFINE VARIABLE dir_name AS CHARACTER NO-UNDO FORMAT "x(64)"
  LABEL "Enter the name of the directory you want to create.".

UPDATE dir_name.
OS-CREATE-DIR VALUE(dir_name).
stat = OS-ERROR.
IF stat NE 0 THEN
  MESSAGE "Directory not created. System Error #" stat.
```

Notes

- The directory name must conform to the naming conventions of the underlying operating system.
- If a specified directory cannot be created, ABL returns an error code.
- Although an error can occur during execution of this statement, the statement does not generate an error message, raise an error condition, or affect the program's flow in any way. Check for an execution error by using the OS-ERROR function and evaluating the return.

See also

OS-ERROR function

OS-DELETE statement

Executes an operating system file or directory delete from within ABL. Can delete one or more files, a directory, or an entire directory branch.

Syntax

```
OS-DELETE
  { filename | VALUE ( expression ) }...
  [ RECURSIVE ]
```

filename

The name of the files or directories to delete. If you specify a directory that is not empty, you must also specify the RECURSIVE option to delete both the files contained within the directory and the directory itself. The file or directory name can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

VALUE (*expression*)

An expression that returns the name of the files or directories to delete. *expression* can contain constants, field names, and variable names. The file or directory name can contain Unicode characters.

RECURSIVE

Instructs OS-DELETE to delete all subdirectories of the directory named in *filename*, as well as the directory itself. Before a directory or subdirectory is deleted, its files are deleted.

Example

This procedure opens a dialog box that prompts the user to choose a file to delete, then uses the OS-DELETE statement to delete the file:

r-os-del.p

```
DEFINE VARIABLE filename  AS CHARACTER NO-UNDO.
DEFINE VARIABLE OKpressed AS LOGICAL   NO-UNDO INITIAL TRUE.

Main:
REPEAT:
  SYSTEM-DIALOG GET-FILE filename
    TITLE "Choose File to Delete"
    MUST-EXIST
    USE-FILENAME
    UPDATE OKpressed.

  IF OKpressed = FALSE THEN LEAVE Main.
  ELSE OS-DELETE VALUE(filename).
END.
```

Notes

- The filenames and directory names must conform to the naming conventions of the underlying operating system.
- If OS-DELETE encounters files or directories that are protected against deletes, it skips over them, generates an error code, but continues to delete any unprotected files and subdirectories that are specified. If several such files or directories are encountered, OS-ERROR returns information on the last error only. If a subdirectory cannot be deleted, then the named directory is not deleted.
- You cannot use wildcard characters to specify files or directories.

- Although an error can occur during execution of this statement, the statement does not generate an error message, raise an error condition, or affect the program's flow in any way. Check for an execution error by using the OS-ERROR function and evaluating the return.

See also

OS-ERROR function

OS-DRIVES function
(Windows only)

Returns a comma-separated list of available drives.

Syntax

```
OS-DRIVES
```

Example

The following procedure populates a selection list with the output of the OS-DRIVES function, and then displays the list and prompts the user to select a drive. The procedure then informs the user that subsequent writes will be to the selected drive.

r-os-drv.p

```
DEFINE VARIABLE drives AS CHARACTER NO-UNDO LABEL "Select a Drive"
  VIEW-AS SELECTION-LIST INNER-CHARS 3 INNER-LINES 5.

DEFINE FRAME f
  drives.

drives:LIST-ITEMS = OS-DRIVES.
UPDATE drives WITH FRAME f.
MESSAGE "Files will be written to drive" INPUT drives:SCREEN-VALUE.
```

Note

On platforms other than Windows, OS-DRIVES compiles and executes, but returns the empty string ("").

OS-ERROR function

Returns, as an INTEGER value, an ABL error code that indicates whether an execution error occurred during the last OS-APPEND, OS-COPY, OS-CREATE-DIR, OS-DELETE, OS-RENAME or SAVE CACHE statement.

Syntax

```
OS-ERROR
```

Example

The following procedure prompts the user to enter a file to delete, attempts to delete the file, and then calls the OS-ERROR function to check for an execution error. If an error occurs, the procedure branches based on the error number and responds accordingly.

r-os-err.p

```
DEFINE VARIABLE err-status AS INTEGER   NO-UNDO.
DEFINE VARIABLE filename   AS CHARACTER NO-UNDO FORMAT "x(40)"
  LABEL "Enter a file to delete".

UPDATE filename.
OS-DELETE VALUE(filename).
err-status = OS-ERROR.

IF err-status <> 0 THEN
CASE err-status:
  WHEN 1 THEN
    MESSAGE "You are not the owner of this file or directory.".
  WHEN 2 THEN
    MESSAGE "The file or directory you want to delete does not exist.".
  OTHERWISE
    DISPLAY "OS Error #" + STRING(OS-ERROR,"99") FORMAT "x(13)"
      WITH FRAME b.
END CASE.
```

Notes

- This function returns 0 if no error occurred.
- Use this function immediately following an OS-APPEND, OS-COPY, OS-CREATE-DIR, OS-DELETE, OS-RENAME, or SAVE CACHE statement to determine whether an error occurred during the statement's execution. If you do not, the next use of one of these statements overwrites the previous error code.
- The following table lists the ABL error codes that the OS-ERROR function can return.

Table 49: ABL OS-ERROR codes

Error number	Description
0	No error
1	Not owner
2	No such file or directory
3	Interrupted system call
4	I/O error
5	Bad file number
6	No more processes
7	Not enough core memory
8	Permission denied

Error number	Description
9	Bad address
10	File exists
11	No such device
12	Not a directory
13	Is a directory
14	File table overflow
15	Too many open files
16	File too large
17	No space left on device
18	Directory not empty
999	Unmapped error (ABL default)

See also

OS-APPEND statement, OS-COPY statement, OS-CREATE-DIR statement, OS-DELETE statement, OS-RENAME statement, SAVE CACHE statement

OS-GETENV function

Returns a string that contains the value of the desired environment variable in the environment in which the ABL session is running.

Syntax

```
OS-GETENV ( environment-variable )
```

environment-variable

The name of the environment variable whose value you want to find.

Example

This procedure prompts a user for a report name. It then builds the full pathname where the report will be stored, using OS-GETENV to find the DLC directory. Finally, the procedure displays the full pathname.

r-os-env.p

```
DEFINE VARIABLE pathname    AS CHARACTER NO-UNDO FORMAT "x(32)"
  LABEL "The report will be stored in ".
DEFINE VARIABLE report_name AS CHARACTER NO-UNDO FORMAT "x(32)"
  LABEL "Please enter report name." .

UPDATE report_name.
pathname = OS-GETENV("DLC") + "/" + report_name.
DISPLAY pathname WITH FRAME b SIDE-LABELS.
```

Notes

- If the environment variable is not defined, this statement returns the Unknown value (?).
- This function returns the value of an environment variable defined before the ABL session started, not a variable defined during the session.
- Since environment variables are case sensitive in some environments, make sure that the name you supply is the correct case.
- For security reasons some operating systems may not return certain system environment variable values. In these cases the Unknown value (?) is returned.

OS-RENAME statement

Executes an operating system file rename or directory rename command from within ABL.

Syntax

```
OS-RENAME
  { source-filename | VALUE ( expression ) }
  { target-filename | VALUE ( expression ) }
```

source-filename

The name of the file or directory to rename. The file or directory name can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

VALUE (*expression*)

An expression that returns the name of the file or directory to rename. *expression* can contain constants, field names, and variable names. The file or directory name can contain Unicode characters.

target-filename

The new name of the file or directory. The file or directory name can contain Unicode characters.

VALUE (*expression*)

An expression that returns the new name of the file or directory. *expression* can contain constants, field names, and variable names. The file or directory name can contain Unicode characters.

Example

This procedure opens a dialog box that prompts the user to choose a file to rename. It then prompts for a new name. Finally, the procedure uses the OS-RENAME statement to rename the file.

r-os-nam.p

```
DEFINE VARIABLE sourcefile AS CHARACTER NO-UNDO.
DEFINE VARIABLE targetfile AS CHARACTER NO-UNDO FORMAT "x(20)"
  VIEW-AS FILL-IN.
DEFINE VARIABLE OKpressed  AS LOGICAL   NO-UNDO INITIAL TRUE.

Main:
REPEAT:
  SYSTEM-DIALOG GET-FILE sourcefile
    TITLE "Choose a File or Directory to Rename"
    MUST-EXIST
    USE-FILENAME
    UPDATE OKpressed.
  IF OKpressed = FALSE THEN
    LEAVE Main.
  UPDATE targetfile WITH FRAME newnameframe.
  OS-RENAME VALUE(sourcefile) VALUE(targetfile).
END.
```

Notes

- The filenames or directory names must conform to the naming conventions of the underlying operating system.
- If *source-filename* and *target-filename* specify different directories, this statement both renames the file and moves it to the new directory.
- Although an error can occur during execution of this statement, the statement does not generate an error message, raise an error condition, or affect the program's flow in any way. Check for an execution error by using the OS-ERROR function and evaluating the return.

See also

OS-ERROR function

OUTPUT CLOSE statement

Closes the default output destination or the output stream you name with the STREAM keyword in a prior OUTPUT TO statement.

Syntax

```
OUTPUT [ STREAM stream | STREAM-HANDLE handle ] CLOSE
```

`STREAM` *`stream`*

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream. See the DEFINE STREAM statement reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams.

`STREAM-HANDLE` *`handle`*

Specifies the handle to a stream. If *`handle`* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

Example

This procedure sends Customer data to a file by using the OUTPUT TO statement. All statements that normally send output to the terminal send output to the file named `cust.dat`. After all Customer data is written to the file, the OUTPUT CLOSE statement resets the output destination, usually the terminal. The final DISPLAY statement displays "`Finished`" on the terminal.

r-out.p

```
OUTPUT TO cust.dat.

FOR EACH Customer NO-LOCK:
  DISPLAY Customer.CustNum Customer.Name Customer.Address Customer.Address2
    Customer.City Customer.State Customer.Country SKIP(2)
    WITH 1 COLUMN SIDE-LABELS.
END.

OUTPUT CLOSE.

DISPLAY "Finished".
```

Notes

- The default output destination is the destination that was active when the procedure began. The output destination is usually the terminal unless the current procedure was called by another procedure while a different destination was active.
- A form feed (new page) is automatically output when a PAGED output stream is closed.
- If the output destination is the Windows clipboard, this statement writes all buffered output data to the clipboard in CF-TEXT format and clears the buffer.
- For more information on directing output, see *OpenEdge Programming Interfaces*.

See also

DEFINE STREAM statement, OUTPUT TO statement, Stream object handle

OUTPUT THROUGH statement

Identifies a new output destination as the input to a process that the AVM starts.

Syntax

```
OUTPUT [ STREAM stream | STREAM-HANDLE handle ] THROUGH
  { program-name | VALUE ( expression ) }
  [ argument | VALUE ( expression ) ] ...
  [ ECHO | NO-ECHO ]
  [ MAP protermcap-entry | NO-MAP ]
  [ PAGED ]
  [ PAGE-SIZE {constant | VALUE ( expression ) } ]
  [ UNBUFFERED ]
  [
      NO-CONVERT
    | { CONVERT
          [ TARGET target-codepage ]
          [ SOURCE source-codepage ]
      }
  ]
```

STREAM *stream*

The name of a stream. If you do not name a stream, the AVM uses the unnamed stream. See the DEFINE STREAM statement reference entry and topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams.

STREAM-HANDLE *handle*

Specifies the handle to a stream. If *handle* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

program-name

The name of the program to which you are supplying data from a procedure. This can be a standard command or your own program. The name can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

VALUE (*expression*)

An expression whose value is the name of a UNIX program to which you are supplying data from a procedure. The program name can contain Unicode characters. An expression is also the argument that you want to pass to the UNIX program. OUTPUT THROUGH passes the value of *expression* as a character string.

argument

An argument you want to pass to the UNIX program. The OUTPUT THROUGH statement passes this argument as a character string.

If the argument is the literal value paged, page-size, echo, no-echo, or unbuffered, you must enclose it in quotes to prevent the AVM from using that argument as one of the PAGED, PAGE-SIZE, ECHO, NO-ECHO, or UNBUFFERED options for the OUTPUT THROUGH statement.

ECHO

Sends all input data read from a file to the UNIX program. The AVM echoes data by default.

NO-ECHO

Suppresses the echoing of input data to the UNIX program.

MAP *protermcap-entry* | NO-MAP

The *protermcap-entry* is an entry from the PROTERMCAP file. Use MAP to send output to a device that requires different character mappings than those in effect for the current output stream. Typically, *protermcap-entry* is a slash-separated combination of a standard device entry and one or more language-specific add-on entries (MAP laserwriter/french or MAP hp2/spanish/italian, for example). The AVM uses the PROTERMCAP entries to build a translation table for the stream. Use NO-MAP to make the AVM bypass character translation altogether. See *Manage ABL Applications* for more information on PROTERMCAP. See *Internationalize ABL Applications* for more information on national language support.

PAGED

Formats the output into pages.

PAGE-SIZE { *constant* | VALUE (*expression*) }

Specifies the number of lines per page. The *expression* is a constant, field name, variable name, or expression whose value is an integer. The default number of lines per page is 56. If you use the TERMINAL option to direct output to the terminal, the default number of lines per page is the number of lines of TEXT widgets that fit on the screen. If you specify a non-zero value for PAGE-SIZE, then the PAGED option is assumed. If you specify PAGE-SIZE 0, the output is not paged.

UNBUFFERED

Writes one character at a time to a normally buffered data source, such as a file. Use the UNBUFFERED option only when you can intermingle your UNIX output (with the ABL UNIX statement) and your ABL output (with the OUTPUT THROUGH statement). That is, the OUTPUT THROUGH statement manages the buffering of output between the ABL procedure the UNIX program that it invokes, but it does not handle the buffering of output to any other programs that the ABL procedure might also invoke.

CONVERT

Allows you to modify the character conversions occurring between the UNIX program and ABL. By default, the OUTPUT TO statement converts characters from the code page specified with the Internal Code Page (-cpinternal) parameter to the code page specified with the Stream Code Page (-cpstream) parameter.

If you specify SOURCE *source-codepage* alone, the conversion accepts *source-codepage* as the code page name used in the AVM memory (instead of -cpinternal).

If you specify TARGET *target-codepage*, the conversion accepts *target-codepage* as the code page of the UNIX program (instead of -cpstream).

If you specify both SOURCE *source-codepage* and TARGET *target-codepage*, it converts characters from the *source-codepage* to *target-codepage* (instead of -cpinternal to -cpstream).

TARGET *target-codepage*

Specifies the target code page of the character conversion (replacing -cpstream). The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that ABL uses for character management).

SOURCE *target-codepage*

Specifies the source code page of the character conversion (replacing -cpinternal). The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that ABL uses for character management).

NO-CONVERT

Specifies that no character conversions occur between the external file and ABL. By default, the OUTPUT THROUGH statement converts characters from the -cpinternal code page to the -cpstream code page.

Examples

In this example, the Customer names are displayed. This output is sent as input to the UNIX wc (word count) command. The output of wc is directed to the file wcdata using the standard UNIX redirection symbol (>). Finally, the results are displayed as three integers that represent the number of lines, words, and characters that were in the data sent to wc.

r-othru.p

```
OUTPUT THROUGH wc > wcdata.
/* Word count UNIX utility */

FOR EACH Customer NO-LOCK:
  DISPLAY Customer.Name WITH NO-LABELS NO-BOX.
END.

OUTPUT CLOSE.
PAUSE 1 NO-MESSAGE.
UNIX cat wcdata.
UNIX SILENT rm wcdata.
```

The r-othru2.p procedure uses the UNIX crypt program, which accepts lines of data, applies an algorithm based on an encryption key and writes the result to the UNIX standard output stream, that can be directed to a file. The output from the procedure is directed to crypt, which encrypts the customer names based on the password, mypass. The results of the encryption are stored in the ecust file. Then, the AVM decrypts and displays this file.

r-othru2.p

```
OUTPUT THROUGH crypt mypass > ecust.

FOR EACH Customer NO-LOCK WHERE Customer.CustNum < 10:
  DISPLAY Customer.Name WITH NO-LABELS NO-BOX.
END.

OUTPUT CLOSE.

UNIX crypt mypass <ecust.
```

Notes

- When you use the OUTPUT CLOSE statement to close an output destination used by an OUTPUT THROUGH statement, the AVM closes the pipe, waits one second, and then continues.
- For any character conversions to occur, all of the necessary conversion tables must appear in `convmap.cp` (a binary file that contains all of the tables that ABL uses for character management).
- If you specify a value of "undefined" for either *`source-codepage`* or *`target-codepage`*, no character conversion is performed.
- For more information on output destinations, see *OpenEdge Programming Interfaces*.

See also

DEFINE STREAM statement, OUTPUT CLOSE statement, OUTPUT TO statement, Stream object handle

---

OUTPUT TO statement

Specifies the new output destination for a stream.

Syntax

```
OUTPUT [ STREAM stream | STREAM-HANDLE handle ] TO
  {     PRINTER [ printer-name ]
      | opsys-file
      | opsys-device
      | TERMINAL
      | VALUE ( expression )
      | "CLIPBOARD"
  }
  [ LOB-DIR { constant | VALUE ( expression ) } ]
  [ NUM-COPIES { constant | VALUE ( expression ) } ]
  [ COLLATE ]
  [ LANDSCAPE | PORTRAIT ]
  [ APPEND ]
  [ BINARY ]
  [ ECHO | NO-ECHO ]
  [ KEEP-MESSAGES ]
  [ NO-MAP | MAP protermcap-entry ]
  [ PAGED ]
  [ PAGE-SIZE { constant | VALUE ( expression ) } ]
  [ UNBUFFERED ]
  [     NO-CONVERT
      | { CONVERT
            [ TARGET target-codepage ]
            [ SOURCE source-codepage ]
        }
  ]
```

`STREAM` *stream*

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream. See the DEFINE STREAM statement reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams.

`STREAM-HANDLE` *handle*

Specifies the handle to a stream. If *handle* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the chapter on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

`PRINTER [` *printer-name* `]`

By default, this option sends output to the printer defined in the default print context. Specify a printer name to send output to a specific printer. Specifying a printer name overrides, but does not change, the printer defined in the default print context.

When you use this option, it implies that the device you are sending output to is paged, unless you also specify PAGE-SIZE 0.

In Windows, you must specify network printers in Universal Naming Convention format. For example:

```
\\fs_dev\hplas4
```

On UNIX, the printer spooling facilities (`lp` or `lpr`) are used automatically.

opsys-file

The absolute or relative pathname of a file to which you want to direct output. If you specify a relative pathname, the AVM locates the pathname relative to the current working directory. The pathname can contain up to 255 characters. If a file with the specified pathname already exists, the AVM overwrites it. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

opsys-device

Represents the name of an operating system device.

TERMINAL

Indicates that you want to direct output to the terminal. The terminal is the default output destination.

VALUE (*expression*)

Represents an expression whose value is the destination to which you want to send data. The value can contain Unicode characters.

"CLIPBOARD"(graphical interfaces only)

Specifies the system clipboard as the output destination. The quotes are required.

LOB-DIR { *constant* | VALUE (*expression*) }

Specifies the directory in which you want the EXPORT statement to write large object data files (such as BLOB and CLOB data files). The *constant* and *expression* arguments are character expressions that evaluate to an absolute pathname or a relative pathname (relative to the directory specified in *opsys-file*).

If the specified character expression evaluates to either the Unknown value (?) or a directory that does not exist, or you do not have permission to write to the specified directory, the AVM raises the ERROR condition.

The LOB-DIR option is valid only when you specify an operating system file as the output destination.

NUM-COPIES { *constant* | VALUE (*expression*) }

Specifies the number of copies to print. The *constant* or *expression* parameters must evaluate to a positive integer. This option is supported in Windows only, and only with printer drivers that support multi-copy printing. Specifying the number of copies to print overrides, but does not change, the number of copies defined in the default print context.

The following statement prints three copies of each output page on the selected printer:

```
OUTPUT TO PRINTER NUM-COPIES 3.
```

COLLATE

Specifies whether multiple copies of output pages print in collated order. This option is supported in Windows only, and only with printer drivers that support collation.

LANDSCAPE

Specifies a landscape page orientation. This option is supported in Windows only, and only with printer drivers that support landscape page orientation. Specifying a page orientation overrides, but does not change, the page orientation defined in the default print context.

The following statement prints three copies of each output page with a landscape orientation on the selected printer:

```
OUTPUT TO PRINTER LANDSCAPE NUM-COPIES 3.
```

PORTRAIT

Specifies a portrait page orientation. This option is supported in Windows only, and only with printer drivers that support portrait page orientation. Specifying a page orientation overrides, but does not change, the page orientation defined in the default print context.

APPEND

Appends the output to the end of a file.

BINARY

Allows output to be written directly without any conversion or interpretation.

ECHO

Sends all input data read from a file to the output destination. Data is echoed by default.

NO-ECHO

Suppresses the echoing of input data to the output destination.

KEEP-MESSAGES

Causes the following messages not to echo to the default window: ABL error and warning messages, and messages from the MESSAGE statement. If you specify KEEP-MESSAGES, these messages are sent only to the output stream you specify.

MESSAGE statements using the VIEW AS ALERT BOX option are an exception. The KEEP-MESSAGES option does not suppress the message box if there is a way available to display it.

MAP *protermcap-entry* | NO-MAP

The *protermcap-entry* value is an entry from the PROTERMCAP file. Use MAP to send output to a device that requires different character mappings than those in effect for the current output stream. Typically, *protermcap-entry* is a slash-separated combination of a standard device entry and one or more language-specific add-on entries (MAP laserwriter/french or MAP hp2/spanish/italian, for example). The AVM uses the PROTERMCAP entries to build a translation table for the stream. Use NO-MAP to make the AVM bypass character translation altogether. See *Manage ABL Applications* for more information on PROTERMCAP. See *Internationalize ABL Applications* for more information on national language support.

PAGED

Formats the output into pages. Form feeds are represented by ^L (**CTRL+L**). When output is PAGED, a page break occurs every 56 lines. PAGED is automatic for output to a printer.

PAGE-SIZE { *constant* | VALUE (*expression*) }

Specifies the number of lines per page. The *expression* is a constant, field name, variable name, or expression whose value is an integer. The default number of lines per page is 56. If you are using the TERMINAL option to direct output to the terminal, the default number of lines per page is the number of lines of TEXT widgets that fit in the window. If you specify a non-zero value for *n*, then the PAGED option is assumed. If you specify PAGE-SIZE 0, the output is not paged in character mode; in a graphical interface, the default page size is used.

UNBUFFERED

Writes one character at a time to a normally buffered data source, such as a file. Use the UNBUFFERED option only when you can intermingle your UNIX output (with the ABL UNIX statement) and your ABL output (with the OUTPUT TO statement). That is, the OUTPUT TO statement manages the buffering of output between the ABL procedure the UNIX program that it invokes, but it does not handle the buffering of output to any other programs that the ABL procedure might also invoke.

CONVERT

Allows you to modify the character conversions occurring between the external file and memory. By default, the OUTPUT TO statement converts characters from the code page specified with the Internal Code Page (-cpinternal) parameter to the code page specified with the Stream Code Page (-cpstream) parameter . If you specify SOURCE *source-codepage* alone, the conversion accepts *source-codepage* as the code page name used in memory (instead of -cpinternal). If you specify TARGET *target-codepage*, the conversion accepts *target-codepage* as the code page of the external file (instead of -cpstream). If you specify both SOURCE *source-codepage* and TARGET *target-codepage*, it converts characters from the *source-codepage* to *target-codepage* (instead of -cpinternal to -cpstream).

TARGET *target-codepage*

Specifies the target code page of the character conversion (replacing -cpstream). The name that you specify must be a valid code page name available in the `DLC/convmap.cp` file (a binary file that contains all of the tables that ABL uses for character management).

SOURCE *target-codepage*

Specifies the source code page of the character conversion (replacing -cpinternal). The name that you specify must be a valid code page name available in the `DLC/convmap.cp` file (a binary file that contains all of the tables that ABL uses for character management).

NO-CONVERT

Specifies that no character conversions occur between the external file and memory. By default, the OUTPUT statement converts characters from the -cpinternal code page to the -cpstream code page.

Examples

The r-out.p procedure sends Customer data to a file. The OUTPUT TO statement directs subsequent output to a file, so all statements that normally send output to the terminal send output to that file. After all the Customer data has been displayed to the file, the OUTPUT CLOSE statement resets the output destination to its previous state, usually the terminal. The final DISPLAY statement displays Finished on the terminal because that is the new output destination.

r-out.p

```
OUTPUT TO cust.dat.

FOR EACH Customer NO-LOCK:
  DISPLAY Customer.CustNum Customer.Name Customer.Address Customer.Address2
    Customer.City Customer.State Customer.Country SKIP(2)
    WITH 1 COLUMN SIDE-LABELS.
END.

OUTPUT CLOSE.
DISPLAY "Finished".
```

The r-termpg.p procedure sends Customer data to the terminal. The OUTPUT TO TERMINAL PAGED statement directs output to the terminal in a paged format; all statements send output to the terminal one page at a time.

r-termpg.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

OUTPUT TO TERMINAL PAGED.

FOR EACH Customer NO-LOCK BREAK BY Customer.SalesRep:
  FIND SalesRep OF Customer NO-LOCK.
  FORM HEADER TODAY
    "Customer Listing For " TO 43
    "Page " TO 55 PAGE-NUMBER - ix  TO 58 FORMAT "99"
    (SalesRep.RepName) FORMAT "x(30)" AT 25
    WITH FRAME hdr PAGE-TOP CENTERED.
  VIEW FRAME hdr.
  DISPLAY Customer.CustNum COLUMN-LABEL "Customer!Number"
    Customer.Name LABEL "Name"
    Customer.Phone COLUMN-LABEL "Phone!Number" WITH CENTERED.
  IF LAST-OF(Customer.SalesRep) THEN DO:
    ix = PAGE-NUMBER.
    PAGE.
  END.
END. /* FOR EACH Customer */

OUTPUT CLOSE.
```

Notes

- Any ABL statement or construct with the option to output data to a stream can write to the stream specified by this statement.

- The OUTPUT TO TERMINAL statement is the default unless the procedure was called by another procedure while a different output destination was active. The output destination at the beginning of the procedure is the current output destination of the calling procedure.
- The OUTPUT TO TERMINAL PAGED statement clears the screen and displays output on scrolling pages the length of the screen. The AVM pauses before each page header. You can alter the pause using the PAUSE statement.
- ABL can display paged output to the terminal for frames that are wider than the width of the screen. The output is wrapped.
- To send output to a file correctly, you must specify the STREAM-IO option of the Frame phrase for any frame you use to write the file.
- If you send data to a file and you plan to use that data file later as input to a procedure, consider using the EXPORT statement. See the INPUT FROM statement reference entry for more information.
- If you send output to a device other than the terminal, ROW options in Frame phrases have no effect. ROW options also have no effect when you send output to a PAGED terminal. If you do not use the NO-BOX option with a Frame phrase, the AVM omits the bottom line of the box, converts the top line to blanks, and ignores the sides of the box.
- All messages, including Compiler error messages and messages produced by the MESSAGE statement, are sent to the current output destination.
- If the field being output is MEMPTR, you must use the BINARY and NO-CONVERT mode of operation to prevent your data from becoming corrupted if it contains binary data.
- With the BINARY and NO-CONVERT options, you will not get a translation of new-lines to the appropriate characters for your operating system and there will be no code page conversion between `-cpinternal` and `-cpstream`.
- If the field being output is MEMPTR and your MEMPTR contains ASCII data you may want code page conversion. However, you cannot get conversion by using the CONVERT parameter on the MEMPTR. You can get code page conversion by using the MEMPTR with the GET-STRING and CODEPAGE-CONVERT functions and the PUT-STRING statement.
- On UNIX, if you want to use a print spooler with spooler options, you can use the Printer (`-o`) startup parameter to specify the options. See *Startup Command and Parameter Reference* for more information on the Printer startup parameter.
- You must use a printer control sequence to change the number of lines per page produced by your printer.
- Unless otherwise specified, the OUTPUT TO PRINTER statement uses the default print context to determine the printer name, number of copies, and page orientation for a print job. If there is no default print context, the AVM uses the printer control settings from the current environment.
- Use the SYSTEM-DIALOG PRINTER-SETUP statement to let users change the default print context through the Windows Print dialog box.
- Use the PRINTER-NAME attribute of the SESSION system handle to set the printer name in the default print context without user intervention.
- In Windows, the OUTPUT TO statement uses the PrinterFont settings in the current environment (either the Registry, or the [Startup] section of the initialization file) to define a font for a print job. The PrintFont settings are similar to the Font settings in the environment and take the following form:

```
PrinterFont [ n ] = facename
   [ , size = screen-point-size ]
```

OUTPUT TO PRINTER uses the PrinterFont setting. OUTPUT TO LPT*n* uses the corresponding PrinterFont*n* entry. The *facename* parameter in a PrinterFont setting represents any valid Windows font supported on your system. If you specify a font that your printer does not support, printing might take a long time and yield unexpected results. The *screen-point-size* setting represents the point size, in screen units, for the font. The AVM converts the point size to logical printer units.

- OUTPUT TO PRINTER in Windows performs the following processing:
  - Checks the default print context. If there is no default print context, the AVM checks the Windows printer control settings from the current environment. If no printer controls are set, the AVM displays an error message and terminates the print operation.
  - Checks the current environment (either the Registry, or the [Startup] section of the initialization file) for a PrinterFont setting. If there is a valid PrinterFont setting, the AVM uses the font specified for the print job. If there is no PrinterFont setting or the setting specifies a non-existent font, the AVM uses the default printer font for the job. If there is no point size specified for the font in the PrinterFont setting, the AVM uses the default size for the printer.
- OUTPUT TO LPT*n* in Windows performs the following processing:
  - Checks the ports settings in Windows for a definition of the specified LPT port. If there is no definition of the specified port, the AVM displays an error message and terminates the print operation. If multiple definitions exist for a port, the AVM uses the first definition that it finds.
  - Checks the current environment (either the Registry, or the [Startup] section in the initialization file) for a corresponding PrinterFont*n* setting (PrinterFont1 is for LPT1, etc.). If there is a valid corresponding PrinterFont*n* setting, the AVM uses the font specified for the print job. If there is no corresponding PrinterFont*n* setting or the setting specifies a non-existent font, the AVM uses the "courier new" font for the job and calculates the font height to fit 60 lines on a page. If there is no point size specified for the font in the PrinterFont*n* setting, the AVM uses the default size for the printer.
  - Defines a header at the top of each page in the output. The size of the header is based upon the following calculation: 1.5 * *font-height*.
- In Windows only, OUTPUT TO "CLIPBOARD" buffers all output to the specified stream until the next OUTPUT CLOSE for that stream. The OUTPUT CLOSE statement then writes the output to the Windows clipboard in CF-TEXT format. You can buffer only up to 64K of data between any stream-related pair of OUTPUT TO "CLIPBOARD" and OUTPUT CLOSE statements. Any additional buffered data is lost.

  For information on providing additional clipboard reading and writing capabilities to your application, see *OpenEdge Programming Interfaces* and the CLIPBOARD system handle reference entry.
- For any character conversions to occur, all of the necessary conversion tables must appear in `convmap.cp` (a binary file that contains all of the tables that ABL uses for character management).
- If you specify a value of "undefined" for either *source-codepage* or *target-codepage*, no character conversion is performed.
- The OpenEdge ADE toolset provides a portable solution for printing text files. The solution is a procedure called `_osprint.p` and it is located in the `adecomm` directory in the OpenEdge product directory (`DLC`). The `_osprint.p` procedure sends a specified text file to the default printer as paged output. For more information on the `_osprint.p` procedure, see *OpenEdge Programming Interfaces*.
- For more information on changing your output destination, see *OpenEdge Programming Interfaces*.

See also

CLIPBOARD system handle, DEFINE STREAM statement, INPUT-OUTPUT CLOSE statement, PAGE-SIZE function, SESSION system handle, Stream object handle, SYSTEM-DIALOG PRINTER-SETUP statement

OVERLAY statement

Inserts content from a specified expression into a field or variable replacing existing characters, bytes, or columns.

Syntax

```
OVERLAY ( target , position [ , length [ , type ] ] ) = expression
```

target

The name of a CHARACTER field or variable, or a LONGCHAR variable, that you want to overlay with the *expression*.

position

An integer expression that indicates the first position in the *target* where you want to overlay the *expression*. The value of the *position* must be positive. If the *position* is longer than the *target*, ABL pads the *target* with blanks to match the *position*.

length

An integer expression that indicates the number of positions you want to allocate for the storage of the *expression*. The *expression* is truncated or padded with blanks to match *length*. If you do not use the *length* argument or specify -1 as the *length*, OVERLAY uses the entire *expression*.

type

A character expression that directs ABL to interpret the specified *position* and *length* values as character units, bytes, or columns. A double-byte character registers as one character unit. By default, ABL interprets the specified *position* and *length* values as character units.

There are three valid types: "CHARACTER," "RAW," and "COLUMN." The expression "CHARACTER" specifies character units. The expression "RAW" specifies bytes. The expression "COLUMN" specifies display or print character-columns. If you specify the type as a constant expression, ABL validates the type specification at compile time. If you specify the type as a non-constant expression, the AVM validates the type specification at run time.

Note: If the *target* is a LONGCHAR expression, "CHARACTER" is the only valid type.

expression

A CHARACTER or LONGCHAR expression that results in a character string you want to overlay on the *target*. If you specify the *length*, the *expression* is truncated or padded with blanks to match the *length.*

Example

The `r-replc1.p` procedure lets you search for, and replace text strings in a paragraph in a window. When you run the procedure, you see the paragraph, which is an array with an extent of five. You also see a prompt. Enter the text string you want the system to search for, and the new text you want in its place. The procedure searches the paragraph, one line at a time, for the text you entered. The procedure uses the OVERLAY statement to replace the string of old text with the string of new text. The procedure also determines the length of the old text and the new text.

r-replc1.p

```
DEFINE VARIABLE chktext  AS CHARACTER NO-UNDO.
DEFINE VARIABLE ix       AS INTEGER   NO-UNDO.
DEFINE VARIABLE chkndx   AS INTEGER   NO-UNDO.
DEFINE VARIABLE ndx      AS INTEGER   NO-UNDO.
DEFINE VARIABLE old-text AS CHARACTER NO-UNDO.
DEFINE VARIABLE new-text AS CHARACTER NO-UNDO.
DEFINE VARIABLE max-len  AS INTEGER   NO-UNDO.
DEFINE VARIABLE comment  AS CHARACTER NO-UNDO FORMAT "x(49)" EXTENT 5
  INITIAL ["You are probably interested in OpenEdge because",
    "you have a lot of information to organize.  You",
    "want to get at the information, add to it, and",
    "change it, without a lot of work and aggravation.",
    "You made the right choice with OpenEdge." ].

DISPLAY comment WITH CENTERED FRAME comm NO-LABELS
  TITLE "Why You Chose OpenEdge" ROW 4.

REPEAT:
  SET old-text LABEL "Enter text to search for"
    new-text LABEL "Enter text to replace with"
    WITH FRAME replace SIDE-LABELS CENTERED.

  max-len = MAXIMUM(LENGTH(old-text), LENGTH(new-text)).
  DO ix = 1 TO 5:
    ndx = 1.
    DO ndx = 1 TO LENGTH(comment[ix]):
      chktext = SUBSTRING(comment[ix], ndx).
      chkndx = INDEX(chktext, old-text).
      IF chkndx <> 0 THEN DO:
        ndx = ndx + chkndx - 1.
        OVERLAY(comment[ix], ndx, max-len, "CHARACTER") = new-text.
        ndx = max-len.
      END.
    END.
    DISPLAY comment[ix] WITH FRAME comm.
  END.
END.
```

Notes

- The SUBSTRING and OVERLAY statements use the *length* option differently. For both, the *length* indicates how much of the *target* to replace. However, SUBSTRING always inserts the full *expression* and never pads the *expression* to match the *length*. By contrast, the *length* in OVERLAY determines how much ABL adds to the *target*, even if ABL must truncate the *expression* or pad it with spaces. See the r-sub-over.p procedure for an illustration the differences between these two statements.
- Do not split double-byte characters. This statement allows you to overlay either the lead or trail-byte of the target string when you specify "RAW" as the *type* parameter.

See also

SUBSTRING function, SUBSTRING statement

PAGE statement

Starts a new output page for PAGED output. No action is taken if output is already positioned at the beginning of a page.

Syntax

```
PAGE [ STREAM stream | STREAM-HANDLE handle ]
```

STREAM *stream*

Specifies the name of a stream where output is paged. If you do not name a stream, the AVM uses the unnamed stream.

STREAM-HANDLE *handle*

Specifies the handle to a stream where output is paged. If *handle* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed stream.

Example

This procedure prints a customer report, categorized by state, and starts a new page for each state:

r-page.p

```
DEFINE VARIABLE laststate AS CHARACTER NO-UNDO.

OUTPUT TO PRINTER.
FOR EACH Customer NO-LOCK BY Customer.State:
  IF Customer.State <> laststate THEN DO:
    IF laststate <> "" THEN PAGE.    laststate = Customer.State.
  END.
  DISPLAY Customer.CustNum Customer.Name Customer.Address Customer.City Customer.State.
END.
```

Notes

- If the current output destination is not a paged device (you did not use the PAGED option in the OUTPUT TO statement), the PAGE statement has no effect.
- PAGE has no effect if you are already at the top of a new page.
- If any PAGE-TOP or PAGE-BOTTOM frames are active, they are output prior to the next display.
- For more information on streams and stream handles, see the topics on alternate I/O sources in *OpenEdge Programming Interfaces*.

See also

DEFINE STREAM statement, OUTPUT TO statement, Stream object handle

PAGE-NUMBER function

Returns the page number of the output destination as an INTEGER value. If the output stream is not paged, PAGE-NUMBER returns a value of 0.

Syntax

```
PAGE-NUMBER [ ( stream | STREAM-HANDLE handle ) ]
```

stream

The name of an output stream. If you do not name a stream, PAGE-NUMBER returns the page number of the default unnamed output stream.

STREAM-HANDLE *handle*

The handle to an output stream. If *handle* it is not a valid handle to a stream, the AVM generates a run-time error.

Example

This procedure creates a customer report with a page number on each page:

r-pgnbr.p

```
OUTPUT TO pagenum.txt PAGED.

FOR EACH Customer NO-LOCK:
  FORM HEADER "Customer report" AT 30
    "Page:" AT 60 PAGE-NUMBER FORMAT ">>9" SKIP(1).
  DISPLAY Customer.CustNum Customer.Name Customer.Address Customer.City
    Customer.State Customer.Country.
END.
```

See also

DEFINE STREAM statement, OUTPUT TO statement, PAGE statement, Stream object handle

PAGE-SIZE function

Returns the page size (lines per page) of an output destination as an INTEGER value. If the output stream is not paged, PAGE-SIZE returns a value of 0.

Syntax

```
PAGE-SIZE [ ( stream | STREAM-HANDLE handle ) ]
```

stream

The name of an output stream. If you do not name a stream, PAGE-SIZE returns the page size of the default unnamed output stream.

STREAM-HANDLE *handle*

The handle to an output stream. If *handle* it is not a valid handle to a stream, the AVM generates a run-time error.

Example

This procedure prints a customer report categorized by state. At the end of each state category, it tests to see if there are at least four lines left on the page. The LINE-COUNTER function returns the current line number of output. If that number plus four is greater than the total number of lines on the page (returned by the PAGE-SIZE function), then the procedure skips to a new page. If there are four or more lines left, the procedure skips a line before printing the next customer record.

r-pgsize.p

```
OUTPUT TO PRINTER.

FOR EACH Customer NO-LOCK BREAK BY Customer.State:
  DISPLAY Customer.CustNum Customer.Name Customer.Address Customer.City
    Customer.State.
  IF LAST-OF(Customer.State) THEN DO:
    IF LINE-COUNTER + 4 > PAGE-SIZE THEN PAGE.
    ELSE DOWN 1.
  END.
END.
```

See also

DEFINE STREAM statement, OUTPUT THROUGH statement, OUTPUT TO statement, Stream object handle

Parameter definition syntax

Defines one or more run-time parameters for a user-defined function, a class method (including a constructor), or a class event defined with an ABL method signature. ABL implements both scalar and array parameters for user-defined functions and methods of a class as NO-UNDO variables.

Syntax

```
( parameter [ , parameter ] ... )
```

Note: To define run-time parameters for an ABL procedure, Windows dynamic link library (DLL) routine, UNIX shared library routine, or ActiveX control event procedure, see the DEFINE PARAMETER statement reference entry.

Use the following syntax to define each *parameter*, which can be a primitive, array, class, interface, temp-table, ProDataSet, or buffer:

```
[{ INPUT | OUTPUT | INPUT-OUTPUT }]
{   parameter-name AS {primitive-type-name | [ CLASS ]object-type-name}
  | { LIKE field }
  [ EXTENT [ constant ] ]
  | TABLE temp-table-name [ APPEND ] [ BIND ] [ BY-VALUE ]
  | TABLE-HANDLE temp-table-handle [ APPEND ] [ BIND ] [ BY-VALUE ]
  | DATASET dataset-name [ APPEND ] [ BIND ] [ BY-VALUE ]
  | DATASET-HANDLE dataset-handle [ APPEND ] [ BIND ] [ BY-VALUE ]
}
```

```
BUFFER buffer-name FOR table-name[ PRESELECT ]
```

INPUT | OUTPUT | INPUT-OUTPUT

Specifies the parameter mode, which determines the direction that data travels when the parameter is passed at run time. An INPUT parameter travels from the caller (source), which sets its value, to the called function or method (destination or receiver), which can use the value. An OUTPUT parameter travels from the called function or method (source), which sets its value, back to the caller (destination or receiver), which can use the value. An INPUT-OUTPUT parameter travels both ways, first from the caller, which sets its value; then to the called function or method, which can use and reset the value and pass it back to the caller, which can use the returned value. If a parameter mode is not specified, then the default mode is INPUT.

parameter-name

Specifies the name of a primitive, class, interface, or array parameter.

AS *primitive-type-name*

Specifies a built-in primitive type for the parameter you are defining. The built-in data type (*primitive-type-name*) can be one of the following:

```
CHARACTER | COM-HANDLE | DATE | DATETIME | DATETIME-TZ | DECIMAL
| HANDLE | INT64 | INTEGER | LOGICAL | LONGCHAR | MEMPTR | RAW
| RECID | ROWID | AS-data-type
```

If you are defining a parameter for a user-defined function, a pure ABL class method (without reference to .NET), or a class event defined with an ABL method signature, *AS-data-type* does not specify a valid primitive type name.

If you are defining a parameter for a method that overrides a .NET super class method (abstract or otherwise) or implements a method defined in a .NET interface, *primitive-type-name* must specify the exact .NET data type of the corresponding parameter in the overridden or implemented method. For a .NET mapped data type that is a default match for a given ABL primitive type, you must use the default matching ABL data type, as shown in Table 24: Implicit mappings between .NET and ABL data types. (For example, INTEGER indicates a .NET `System.Int32`.) For a .NET mapped data type that is **not** a default match for one of the ABL primitive types, ABL provides a data type keyword (*AS-data-type*) that you must use to explicitly indicate the required .NET data type, as shown in Table 25: Explicit mappings between ABL and .NET data types. (For example, the AS data type, UNSIGNED-BYTE, indicates a .NET `System.Byte`.)

Note: At run time, a method parameter defined using an *AS-data-type* keyword behaves in ABL like the corresponding ABL primitive type shown in Table 25: Explicit mappings between ABL and .NET data types. (For example, an UNSIGNED-BYTE behaves like an INTEGER.)

Also note that when overriding or implementing a .NET array parameter, you must specify the .NET array object type (for example, `"System.Int32[]"` or `"System.Byte[]"`); you **cannot** use an ABL array equivalent (for example, INTEGER EXTENT or UNSIGNED-BYTE EXTENT).

AS [CLASS] { *object-type-name* }

Specifies the parameter you are defining as an object reference to a class instance. This can have the data type of a class or an interface.

object-type-name

Specifies the type name of an ABL or .NET class or interface. Specify an object type name using the syntax described in the Type-name syntax reference entry. With an appropriate USING statement, you can also specify a class or interface name alone, without the qualifying package or namespace.

You cannot directly specify the type name of a .NET mapped object type (such as `System.Int32`). To define a parameter that matches a .NET mapped type, you must define it as the corresponding ABL primitive type (*primitive-type-name*).

If you are defining a parameter for a method that overrides a .NET super class method (abstract or otherwise) or implements a method defined in a .NET interface, *object-type-name* must specify the exact .NET object type of the corresponding parameter in the overridden or implemented method. However, for .NET inner (nested) type, note the difference in the ABL syntax, which replaces the corresponding period (`.`) in the .NET object type with a plus (+) (see the Type-name syntax reference entry).

Also note that when overriding or implementing a .NET array parameter, you must specify the .NET array object type (for example, `"System.Drawing.Point[]"`); you **cannot** use an ABL array equivalent (such as `System.Drawing.Point EXTENT`).

CLASS

If the specified class or interface type name conflicts with an abbreviation for a built-in primitive type name, such as INT for INTEGER, you must specify the CLASS keyword.

For a class or interface return value, the AVM passes an object reference associated with the class or interface, not a class instance itself. For more information on object references, see the Class-based object reference reference entry.

LIKE *field*

Specifies that the parameter you are defining has the same characteristics as the variable, database field, or temp-table field designated by *field*.

The parameter will take on the characteristics of *field*. These characteristics include data type, extent, label, column-label, format, decimals, view-as and case sensitivity. One exception is UNDO, since all method and function parameters are always NO-UNDO.

You can override the extent of *field* by supplying your own EXTENT phrase. You cannot override any other characteristics because the only variable qualifier available in the parameter definition syntax is EXTENT.

If *field* is a variable or temp-table field, then it must be known by the compiler before the user-defined function, a class method, or class event you are defining can be compiled.

If *field* is a database field, then the database must be connected at compile time, but need not be connected at run time. You may optionally qualify a database field name with its database and/or table name.

EXTENT **[** *constant* **]**

Defines the parameter as an array of data elements with a primitive type (specified using the AS *primitive-type-name* option) or an object type (specified using the AS *object-type-name* option). This option can specify an array parameter as either determinate (has a defined number of elements) or indeterminate (has an undefined number of elements). To define a determinate array parameter, specify the EXTENT option with the *constant* argument. This optional argument is an integer value that represents the number of data elements in the array parameter. To define an indeterminate array parameter, specify the EXTENT option without the *constant* argument.

The EXTENT is part of the parameter data type. For more information, see the Type-name syntax reference entry.

For more information about determinate and indeterminate arrays, see Work with one-dimensional arrays in *Basic ABL*.

If you are using the AS option without the EXTENT option, or you specify *constant* as 0, the parameter is not an array parameter.

TABLE *temp-table-name*

Specifies a parameter for a static temp-table.

You can pass a temp-table parameter to both local and remote user-defined functions, as well as to methods of a class. The AVM passes the parameter by value, by default. That is, the caller and the called routine each have their own instance of the temp-table. When you invoke the routine, the AVM deep-copies the parameter from one instance to the other. The table that is copied depends on whether the parameter is INPUT, OUTPUT, or INPUT-OUTPUT. When you pass a temp-table as an INPUT parameter, the AVM replaces the receiving instance with the source instance, by default. You can also append the source instance to the end of the receiving instance by specifying the APPEND option. For more information about the APPEND option, see the option description later in this reference entry.

When passing a temp-table parameter, you can override the default deep copy and pass the parameter by reference or by binding (that is, by passing the parameter using either the BY-REFERENCE or BIND option). Passing a temp-table parameter by reference or by binding allows the caller and the called routine to access the same object instance (instead of deep-copying the parameter).

For more information about passing a temp-table parameter by reference or by binding, see the Parameter passing syntax reference entry. For more information about temp-table parameters, see *ABL Essentials*.

`TABLE-HANDLE` *`temp-table-handle`*

Specifies a temp-table handle parameter.

`DATASET` *`dataset-name`*

Specifies a parameter for a static ProDataSet.

You can pass a ProDataSet parameter to both local and remote user-defined functions, as well as to methods of a class. The AVM passes the parameter by value, by default. That is, the caller and the called routine each have their own instance of the ProDataSet. When you invoke the routine, the AVM deep-copies the parameter from one instance to the other. The table that is copied depends on whether the parameter is INPUT, OUTPUT, or INPUT-OUTPUT. When you pass a ProDataSet as an INPUT parameter, the AVM replaces the receiving instance with the source instance, by default. You can also append the source instance to the end of the receiving instance by specifying the APPEND option. For more information about the APPEND option, see the option description later in this reference entry.

When passing a ProDataSet parameter, you can override the default deep copy and pass the parameter by reference or by binding (that is, by passing the parameter using either the BY-REFERENCE or BIND option). Passing a ProDataSet parameter by reference or by binding allows the caller and the called routine to access the same object instance (instead of deep-copying the parameter).

For more information about passing a ProDataSet object parameter by reference or by binding, see the Parameter passing syntax reference entry. For more information about ProDataSet object parameters, see *Use ProDataSets*.

`DATASET-HANDLE` *`dataset-handle`*

Specifies a ProDataSet handle parameter.

`APPEND`

Specifies whether or not to append the data from a source instance to the receiving instance of a passed temp-table or ProDataSet parameter. To append INPUT parameter data, specify the APPEND option for the parameter in the method or user-defined function definition. To append OUTPUT parameter data, specify the APPEND option for the parameter in the method or function call.

`BIND`

Indicates that a TABLE, TABLE-HANDLE, DATASET, or DATASET-HANDLE parameter binds a reference-only object in one routine to an object instance defined and instantiated in another local routine.

When you define a reference-only object in the calling routine, and you want to bind that object definition to an object instance in the called routine, define the parameter by specifying the BIND option in an INPUT or INPUT-OUTPUT parameter definition. When you define a reference-only object in the called routine, and you want to bind that object definition to an object instance in the calling routine, define the parameter by specifying the BIND option in an OUTPUT parameter definition. In either case, the reference-only object definition remains bound to the object instance until the routine containing the reference-only object definition is deleted or terminates.

Caution: Do not delete the object or routine to which a reference-only object is bound, or you might be left with references to an object that no longer exists.

You can bind multiple reference-only object definitions to the same object instance. You can also bind a single reference-only object definition to the same object instance multiple times without generating an error. However, you cannot bind a single reference-only object definition to multiple object instances.

When passing one of these parameters to a remote user-defined function, the AVM ignores the BIND option and deep-copies the parameter based on the specified parameter mode.

For more information about passing these parameters by binding, see the Parameter passing syntax reference entry.

BY-VALUE

Specified for an INPUT, OUTPUT, or INPUT-OUTPUT TABLE, TABLE-HANDLE, DATASET, or DATASET-HANDLE parameter in a called routine, this option forces the parameter to be passed to the local routine by value, which overrides any BY-REFERENCE option in the corresponding routine invocation. For more information on BY-REFERENCE, see the Parameter passing syntax reference entry.

Caution: Using BY-VALUE with a temp-table or ProDataSet parameter is only permitted with procedures. You cannot use BY-VALUE with temp-table and ProDataSet parameters in functions or class methods.

BUFFER *buffer-name* FOR *table-name* [PRESELECT]

Defines a buffer parameter, where *buffer-name* is the name you specify for the buffer and *table-name* is the name of a temp-table or database table to which the buffer is attached. However, if a temp-table name is identical to a database table name, all ABL references to the database table name must use fully-qualified *database-name*.*database-table-name* syntax. A buffer parameter is always INPUT-OUTPUT.

Note: You cannot pass a work table buffer to a buffer parameter.

If you use the PRESELECT option and access the buffer parameter in a DO or REPEAT block, the AVM creates an internal list of the records selected. The PRESELECT option tells the AVM to apply that internal list to the buffer you define.

Note: You cannot invoke a user-defined function remotely if it has one or more buffer parameters.

Notes

- When defining an overloaded method or constructor, the signature must be unique among all overloaded method or constructor definitions, respectively, with the same name. The signature consists of the name of the method or constructor, the number of parameters, and the type and mode of each of its parameters, in order from left to right. The signature of a method does not include the method return type or access mode.

  So, overloading allows you to define multiple methods or constructors for a class definition, where all methods or constructors that have the same name all have different signatures. The signature cannot vary only by access mode or return type. If methods within a class definition differ only by their access modes or return types, ABL raises a compilation error.

- For data types that can have an EXTENT for defining arrays, you can define overloaded methods or constructors that are distinguished only by parameters with the same data type, where:

  - One can have the same parameter defined as a primitive (no EXTENT)

  - One can have the same parameter defined with an indeterminate EXTENT
  - Every additional one can have the same parameter defined with a different EXTENT than all the others

- You can have overloaded methods or constructors defined only with TABLE parameters or only with DATASET parameters as long as the corresponding parameter for each overloaded method or constructor is defined with a different schema from all others defined with the same type.
- TABLE and TABLE-HANDLE parameters count as different data types for overloading, as do DATASET and DATASET-HANDLE parameters. However, because the schema is not known at compile time, you can define only one overloaded method or constructor with a TABLE-HANDLE parameter corresponding to other method or constructor parameters that differ only as TABLE parameters. Similarly, you can define only one overloaded method or constructor with a DATASET-HANDLE parameter corresponding to other overloaded method or constructor parameters that differ only as DATASET parameters.
- At compile time, it might not be possible to identify a unique overloaded method to call that is passed a TABLE-HANDLE or DATASET-HANDLE parameter. Without a schema defined, this parameter type matches any respective TABLE or DATASET parameter, as well as any respective TABLE-HANDLE or DATASET-HANDLE parameter. Where such overloaded alternatives exist, the AVM resolves the correct method to call at run time. For more information, see the Parameter passing syntax reference entry.
- A BUFFER parameter cannot be a REFERENCE-ONLY buffer or a BEFORE-TABLE buffer. Using either of these as a BUFFER type of parameter (as opposed to INPUT or OUTPUT) is a compiler error. Instead of passing the buffer, the buffer's table can be passed by-reference.

See also

CONSTRUCTOR statement, DEFINE EVENT statement, FUNCTION statement, METHOD statement, Parameter passing syntax, SUPER function, Type-name syntax, USING statement

---

Parameter passing syntax

Specifies one or more parameters to pass during invocation of an ABL procedure, a user-defined function, a method of a class (ABL or .NET), a class constructor (ABL or .NET), or the built-in Publish( ) event method or PUBLISH statement to publish class or named events, respectively.

Syntax

```
( parameter [ , parameter ]... )
```

The parameters specified by *parameter* must match in number and order, according to mode and data type, as required by the given procedure, user-defined function, method, or constructor definition. Use the following syntax to specify each *parameter*:

```
[ INPUT | OUTPUT | INPUT-OUTPUT ]
{   parm [ AS data-type ]
  | { {   TABLE temp-table-name
        | TABLE-HANDLE temp-table-handle
        | DATASET dataset-name
        | DATASET-HANDLE dataset-handle
      } [ APPEND ] [ BY-VALUE | BY-REFERENCE | BIND ]
    }
}
```

```
BUFFER buffer
```

[INPUT | OUTPUT | INPUT-OUTPUT]

Specifies the parameter mode. For more information on parameter modes, see the reference entries for the DEFINE PARAMETER statement (for procedures) and the Parameter definition syntax (for user-defined functions, methods, and class events).

Note that for methods and constructors, the parameter mode is optional except for certain overloading conditions. Thus, the default mode depends on the type of invocation, as described in the following table.

Table 50: Default parameter passing mode

Invocation type	Default mode
Procedure	INPUT
User-defined function	Uses the mode defined for the corresponding parameter in the function prototype. For more information on defining prototypes for user-defined functions, see the FUNCTION statement reference entry.

Invocation type	Default mode
Method of a class or class constructor	Uses the mode defined for the corresponding parameter in the method or constructor prototype, depending on overloading requirements. For more information on defining prototypes for class-based methods and constructors, see the METHOD statement and CONSTRUCTOR statement reference entries. If the method is overloaded by a given parameter **only** by mode, you must specify the mode for this parameter in the method call. If you do not do so, ABL raises a compile-time ambiguity error. For example, if a method is overloaded twice by an INTEGER parameter, and the parameter for one method definition has the INPUT mode while the parameter for the other method definition has the OUTPUT mode, you must specify the INPUT keyword if you intend to use the method defined with the INPUT parameter.
Publish( ) event method	Uses the mode defined for the corresponding parameter in the class event definition. For more information on defining class events, see the DEFINE EVENT statement reference entry.

When identifying the parameter mode for a .NET method or constructor, each .NET language uses its own keyword syntax to identify it. The following table shows the C# parameter syntax that corresponds to an ABL parameter specified with a given mode. Note that in C#, the default (no keyword) corresponds to the ABL INPUT mode.

Table 51: C# syntax matching ABL parameter modes

ABL parameter mode	Corresponding C# syntax
`INPUT` *`parm`*	*`parm`*
`OUTPUT` *`parm`*	`out` *`parm`*
`INPUT-OUTPUT` *`parm`*	`ref` *`parm`*

Also note, as with ABL class-based methods, the default parameter passing modes for calling .NET methods and constructors are based on the parameter passing modes defined for the method or constructor prototypes, depending on overloading. Thus, you **must** specify the parameter passing mode for each affected parameter if the method is overloaded **only** by a given parameter's mode.

`parm`

Specifies the argument passed to the parameter. This can be either an ABL data element or Expression, or a .NET data element, depending on the parameter mode. For an INPUT parameter, *`parm`* can be an Expression. For an OUTPUT or INPUT-OUTPUT parameter, *`parm`* can be any of the elements defined for the left side of an Assignment (=) statement. For an INPUT-OUTPUT parameter, any property argument must be both writable and readable.

Note: Syntactic limitations require that none of the following data items can be an OUTPUT or INPUT-OUTPUT parameter: a writable handle attribute, a writable property (ABL or .NET) on an object reference, or a data member (ABL or .NET) on an object reference. The limitation is that OUTPUT and INPUT-OUTPUT parameters cannot have a colon in their syntax. For example, `OUTPUT myObjectRef:WritableProperty` is invalid syntax.

For procedures, the data type of *parm* must be compatible with the data type defined for the parameter. Procedures allow the matching of a wide variety of different data types between the passed parameter and the parameter definition. When it can, the AVM converts the passed value from the source data type to an appropriate value in the destination data type, depending on the direction (parameter mode) of the passed value. For procedures, the AVM checks data type matching and appropriate overflow conditions at run time.

For user-defined functions, methods of a class, class constructors, and the Publish( ) event method, the data types of the passed parameter and the parameter definition are validated by the AVM at compile time and must match exactly, unless they have a widening relationship.

The AVM implicitly converts passed parameter values of certain data types from a narrow data type in the source to a widened data type in the destination, depending on the parameter mode. A widened data type is one that can hold all the values of a narrower data type without loss of data. Widening is supported for three related sets of data types, as shown in the following table, where the arrow (—>) indicates the direction that a value can be passed for the parameter.

Table 52: Data type widening support

Narrower —> Wider		
CHARACTER	–	LONGCHAR
INTEGER	INT64	DECIMAL
DATE	DATETIME	DATETIME-TZ

Thus, ABL supports the widening of data types in the direction that the parameter is defined, according to its mode:

- **INPUT parameters** — The data type of the passed parameter can be narrower than the defined parameter.
- **OUTPUT parameters** — The data type of the passed parameter can be wider than the defined parameter.
- **INPUT-OUTPUT parameters** — Widening is not supported. Because values are passed in both directions, the data type of the passed parameter must exactly match the data type of the defined parameter.

For .NET method parameters, ABL also supports widening relationships between ABL and some .NET data types. For more information, see the notes of this reference entry.

For a *parm* that involves .NET data types, either as the parameter definition or as the argument, the requirements for parameter passing depend on the parameter and argument data types and the type of routine that defines the parameter. The following table shows the possible combinations.

Note: The following table refers to .NET value types, .NET mapped data types, .NET arrays of mapped types, boxing, and unboxing. Boxing and unboxing refer to a .NET mechanism for converting between .NET value types and .NET object types. ABL supports a similar mechanism for converting between ABL primitive or array types and a .NET `System.Object` or compatible array object type. For more information on .NET data types, concepts, and compatibility, see the Data types reference entry.

Table 53: Passing parameters involving .NET types

The following parameter data type . . .	In this routine type . . .	Can take these arguments (*parm*) . . .
.NET mapped data type[43]	.NET routine[44]	The same .NET data type (for example, a .NET property), any corresponding ABL primitive type (as shown in Table 24: Implicit mappings between .NET and ABL data types[45]), and on OUTPUT only, a `System.Object`[46]
.NET mapped data type[47]	ABL method that overrides or implements a .NET method	Any compatible ABL primitive type or any compatible .NET mapped data type (for example, a .NET property) as shown in Table 24: Implicit mappings between .NET and ABL data types[45,48]
.NET array	.NET routine	A compatible .NET or ABL array type[49, 50], and on OUTPUT only, a `System.Array`, `System.Object`, or `Progress.Lang.Object`
.NET array	ABL routine[51, 52]	A compatible .NET array type[50, 53]and on OUTPUT only, a `System.Array`, `System.Object`, or `Progress.Lang.Object`,
`System.Array`	.NET routine	On INPUT, any .NET array object type or compatible ABL array type[54], and on OUTPUT only, a `System.Array`, `System.Object`, or `Progress.Lang.Object`
`System.Array`	ABL routine[51, 52]	On INPUT, any .NET array object type[53], and on OUTPUT only, a `System.Array`, `System.Object`, or `Progress.Lang.Object`
`System.Object`	.NET routine	On INPUT, any .NET object type, an ABL primitive type, or compatible ABL array type[54], and on OUTPUT only, a System.Object or Progress.Lang.Object

The following parameter data type . . .	In this routine type . . .	Can take these arguments (*parm*) . . .
`System.Object`	ABL routine[51, 52]	On INPUT, any .NET object type[53], and on OUTPUT only, a `System.Object` or `Progress.Lang.Object`
Any .NET object type (except those in the previous rows)	.NET routine	Any compatible .NET object type, and on OUTPUT only, a `System.Object` or a `Progress.Lang.Object`
Any .NET object type (except those in the previous rows)	ABL routine[51, 52]	Any compatible .NET object type, and on OUTPUT only, a `System.Object` or a `Progress.Lang.Object`
ABL primitive type[47]	ABL routine (always)[51]	Any compatible ABL primitive type or any compatible .NET mapped data type (for example, a .NET property) as shown in Table 24: Implicit mappings between .NET and ABL data types[45,48]
ABL array	ABL routine (always)[51]	A compatible ABL array type[48]
`Progress.Lang.Object`	ABL routine (always)[51]	On INPUT only, any ABL object and any supported .NET object type except a .NET mapped data type[46]

AS *data-type*

Indicates an explicit mapping between an argument with an ABL primitive type and the parameter of a .NET method or constructor. Assuming that the .NET data type defined for the parameter is compatible with the data type of the ABL argument, *data-type* represents an ABL keyword (*AS data type*) that indicates an alternative .NET data type mapping to identify with the .NET parameter. This option is necessary in the following cases:

- When the .NET method or constructor is overloaded by multiple implicit .NET data type mappings for the passed ABL primitive type and the method you want is not identified by the default match for the passed ABL data type. Thus, by explicitly specifying an AS data type, you can disambiguate the implicit .NET overloadings for the method or constructor.

  Note: You can specify a passed parameter AS data type for a .NET method that is not overloaded. However, you have no need to do so unless the method is overloaded by a given parameter.

- The parameter is defined as a `System.Object`, and you want the .NET value of the passed ABL primitive type to be stored as a .NET mapped data type that is not the default match. For example, you might want the `System.Object` parameter to store an ABL INTEGER value as a `System.Int16` instead of as a `System.Int32` (the default match).

For the a list of available keywords that you can specify for *data-type*, as well as the default matches for ABL primitive types with multiple .NET data type mappings, see Table 25: Explicit mappings between ABL and .NET data types in the Data types reference entry.

[43] A .NET mapped data type is any .NET primitive or object type that ABL maps implicitly to an ABL primitive type. For more information, see the Data types reference entry and Table 24: Implicit mappings between .NET and ABL data types.

[44] A .NET routine can be a .NET method or constructor.

[45] For an ABL or .NET method, the parameter and argument data types must match exactly or map according to the widening rules of the target (on INPUT or OUTPUT) described in this entry. Also note that you cannot pass a System.Object on INPUT to a primitive parameter, because it is a typical narrowing case.

[46] You cannot pass parameters where a Progress.Lang.Object is exchanged with a .NET mapped data type, because ABL does not do intermediate boxing or unboxing to a System.Object in order to exchange the value with a Progress.Lang.Object. However, you can pass parameters, exchanging a Progress.Lang.Object with a System.Object (with casting, as appropriate).

[47] For an ABL routine, you cannot define parameters (or any data elements) directly as a .NET mapped data type (such as, System.Int32). To define parameters that take a .NET mapped type as an argument, you must use the matching ABL primitive type, as shown in Table 24: Implicit mappings between .NET and ABL data types (such as, INTEGER). However, if an ABL method overrides a .NET method or implements a method of a .NET interface, you must define any .NET parameter that is a .NET mapped type using an ABL keyword (AS data type) that explicitly indicates this .NET type. Also note that this parameter functions at run time as its matching ABL primitive type, not the mapped type as defined by .NET. For more information, see the Parameter definition syntax reference entry.

[48] ABL does not automatically box or unbox the value of a .NET System.Object or array object argument passed to a compatible ABL primitive or array parameter of an ABL routine. Also note that you cannot provide a Progress.Lang.Object argument to this parameter on OUTPUT, because an ABL primitive or array, even one with object type elements, is not an object itself and ABL does no boxing to a Progress.Lang.Object.

[49] For an INPUT .NET array parameter of mapped type elements, because ABL does not have a unique primitive type to match each .NET mapped data type, ABL allows you to box any ABL array whose element type implicitly maps to the elements of the target .NET array, as shown in Table 24: Implicit mappings between .NET and ABL data types. For example, if the parameter is defined as a .NET "System.Int16[]" array object type, you can pass another "System.Int16[]" or an ABL INTEGER array to box, but you cannot pass an ABL INT64 array to box. On INPUT, you also cannot pass an ABL primitive array to a "System.Object[]" parameter, because ABL does not do automatic boxing at the element level of an array object.

[50] If the parameter is defined as a .NET value type array (such as, "System.Drawing.Size[]"), the argument must be an identical .NET or ABL array type (such as "System.Drawing.Size[]" or System.Drawing.Size EXTENT). Also, while an INPUT System.Array parameter can take any appropriate .NET or ABL array argument, an INPUT "System.Object[]" parameter cannot take any ABL array (for .NET methods) or .NET array argument whose element type is an ABL primitive or .NET value type.

[51] An ABL routine can be an ABL method, constructor, procedure, or user-defined function.

[52] This includes any ABL method that overrides a .NET method or implements a method of a .NET interface.

[53] ABL does not automatically box or unbox a compatible ABL primitive or array data type passed to a corresponding .NET System.Object or array object parameter of an ABL routine. On INPUT, you can pass an ABL BOX function call taking a compatible ABL primitive or array type as input. On OUTPUT, you can provide a System.Object or compatible .NET array object argument and then convert it to an ABL primitive or array type by assigning or using the UNBOX function.

[54] For a .NET method with an ABL primitive type argument: on INPUT, the AVM boxes the argument value into the default matching .NET object type. With a compatible ABL array argument: on INPUT, the AVM boxes the argument as the corresponding .NET array type.

Note: The AS data types in Table 25: Explicit mappings between ABL and .NET data types represent some different data types than you can specify using the AS option to pass a COM method parameter. For more information on passing COM method parameters, see Syntax for accessing COM object properties and methods.

`TABLE` *`temp-table-name`*

Specifies the name of a static temp-table.

This parameter type can match at compile time with any TABLE parameter with the same schema, or any TABLE-HANDLE parameter. If the matching type is TABLE-HANDLE, a run-time check occurs if the TABLE-HANDLE is not the Unknown value (?) in order to ensure that the run-time schemas match. A parameter of a user-defined function or method of a class is verified at compile time, while a parameter of a procedure is verified at run time.

`TABLE-HANDLE` *`temp-table-handle`*

Specifies a handle to a temp-table. Use a temp-table handle as a parameter for a dynamic temp-table. The full schema definition behind the handle and the contents of the temp-table are passed unless the temp-table SCHEMA-MARSHAL attribute is set to minimize or prevent schema marshalling.

This parameter type can match at compile time with any TABLE or TABLE-HANDLE parameter. If the matching type is TABLE-HANDLE, a run-time check occurs if the TABLE-HANDLE is not the Unknown value (?) in order to ensure that the run-time schemas match. A parameter of a user-defined function or method of a class is verified at compile time, while a parameter of a procedure is verified at run time.

`DATASET` *`dataset-name`*

Specifies the name of a static ProDataSet.

This parameter type can match at compile time with any DATASET parameter with the same schema, or any DATASET-HANDLE parameter. If the matching type is DATASET-HANDLE, a run-time check occurs if the DATASET-HANDLE is not the Unknown value (?) in order to ensure that the run-time schemas match. A parameter of a user-defined function or method of a class is verified at compile time, while a parameter of a procedure is verified at run time.

`DATASET-HANDLE` *`dataset-handle`*

Specifies a handle to a ProDataSet. Use a ProDataSet object handle as a parameter for a dynamic ProDataSet. The full schema definition behind the handle and the contents of the ProDataSet object are passed unless the SCHEMA-MARSHAL attribute for one or more of its temp-tables are set to minimize or prevent schema marshalling.

This parameter type can match at compile time with any DATASET or DATASET-HANDLE parameter. If the matching type is DATASET-HANDLE, a run-time check occurs if the DATASET-HANDLE is not the Unknown value (?) in order to ensure that the run-time schemas match. A parameter of a user-defined function or method of a class is verified at compile time, while a parameter of a procedure is verified at run time.

`BUFFER` *`buffer`*

Specifies the name of a buffer.

`APPEND`

Specifies whether or not to append the source temp-table data to the destination temp-table data. To append OUTPUT parameter data, specify the APPEND option for the parameter in the routine

call. To append INPUT parameter data, specify the APPEND option for the parameter in the routine definition.

BY-VALUE | BY-REFERENCE | BIND

Specifies whether to pass a TABLE, TABLE-HANDLE, DATASET, or DATASET-HANDLE parameter by value, by reference, or by binding. The default is BY-VALUE.

You can pass TABLE, TABLE-HANDLE, DATASET, and DATASET-HANDLE parameters to both local and remote procedures. These parameter types are normally passed by value, by default. That is, the calling routine and the called routine each have their own instance of the object, and the parameter is deep-copied from the calling routine's instance to the called routine's instance.

When passing one of these parameters to a local routine, you can override the default in the calling routine by specifying the BY-REFERENCE or BIND option.

Passing one of these parameters to a local routine using the BY-REFERENCE option allows the calling routine and the called routine to access the same object instance. That is, both routines access the calling routine's instance and ignore the called routine's instance. Since the called routine's object instance is ignored, you should define the static object as reference-only by specifying the REFERENCE-ONLY option in the DEFINE statement for the object.

Passing one of these parameters to a local routine using the BIND option allows the calling routine and the called routine to access the same object instance. You can do this by:

- Binding a reference-only static object defined in one routine to an object instance defined in another routine
- Binding an unknown TABLE-HANDLE or DATASET-HANDLE parameter defined in one routine to an object instance defined in another routine

In the static case, you must define a reference-only object in either the calling routine or the called routine by specifying the REFERENCE-ONLY option in the DEFINE statement for the object. You must also define the parameter by specifying the BIND option in the parameter definition.

When you define a reference-only object in the calling routine and pass it to the called routine using the BIND option, the AVM binds the definition of the object in the calling routine to the object instance in the called routine. When you define a reference-only object in the called routine and receive the object from the calling routine, the AVM binds the definition of the object in the called routine to the object instance in the calling routine. In either case, the reference-only object definition remains bound to the object instance until the routine containing the reference-only object definition is deleted or terminates.

Caution: Do not delete the object or routine to which a reference-only object is bound, or you might be left with references to an object that no longer exists.

You can bind multiple reference-only object definitions to the same object instance. You can also bind a single reference-only object definition to the same object instance multiple times without generating an error. However, you cannot bind a single reference-only object definition to multiple object instances.

When passing one of these parameters to a remote procedure, the AVM ignores the BY-REFERENCE and BIND options and deep-copies the parameter based on the specified parameter mode.

Examples

The following two code fragments show how the AS data type works when calling an overloaded .NET method, in this case the `System.Math:Max( )` method. This static .NET method compares two values of the same data type and returns the largest of the two. The first fragment compiles and runs. It compares the value 50, passed as a `System.Byte` (specified by the AS data type, UNSIGNED-BYTE), with the maximum value of a `System.Byte`, returned by the `System.Byte:MaxValue` data member. The result returned by the `Max( )` method is 255, the maximum `System.Byte` value:

```
DEFINE VARIABLE iVal1   AS INTEGER NO-UNDO INITIAL 50.
DEFINE VARIABLE iVal2   AS INTEGER NO-UNDO.
DEFINE VARIABLE iReturn AS INTEGER NO-UNDO.

iVal2 = System.Byte:MaxValue.
iReturn = System.Math:Max( INPUT iVal1 AS UNSIGNED-BYTE,
                           INPUT iVal2 AS UNSIGNED-BYTE ).
```

The second fragment compiles, but returns a run-time error. Again, it passes the same two values to the `System.Math:Max( )` method, but this time passes them as a `System.SByte` (specified by the AS data type, BYTE). A signed byte parameter cannot hold positive values as large as an unsigned byte. So, passing the maximum value of a `System.Byte` (unsigned byte) as a `System.SByte` (signed byte) causes the `Max( )` method to raise a run-time overflow error:

```
DEFINE VARIABLE iVal1   AS INTEGER NO-UNDO INITIAL 50.
DEFINE VARIABLE iVal2   AS INTEGER NO-UNDO.
DEFINE VARIABLE iReturn AS INTEGER NO-UNDO.

iVal2 = System.Byte:MaxValue.
iReturn = System.Math:Max( INPUT iVal1 AS BYTE,
                           INPUT iVal2 AS BYTE ). /* Run-time error */
```

The following code fragment shows an example of ABL data type widening when passing parameters to a .NET method. This example shows INPUT widening, in this case, passing different ABL data types (INTEGER and INT64) that are acceptable as arguments to a `System.Double` input parameter:

```
DEFINE VARIABLE i4Val  AS INTEGER NO-UNDO.
DEFINE VARIABLE i8Val  AS INT64   NO-UNDO.
DEFINE VARIABLE iRoot1 AS DECIMAL NO-UNDO.
DEFINE VARIABLE iRoot2 AS DECIMAL NO-UNDO.

ASSIGN
  i4Val  = System.Int32:MaxValue
  i8Val  = System.Int64:MaxValue
  iRoot1 = System.Math:Sqrt( INPUT i4Val )
  iRoot2 = System.Math:Sqrt( INPUT i8Val ).
```

For more information on data type widening, see the Notes section later in this reference entry.

Notes

- The requirements for calling and passing parameters to built-in handle methods and COM object methods have unique features not shared in calls to procedures, user-defined functions, and methods of a class or class constructors. For information on calling and passing parameters to built-in handle methods and COM object methods, see the information on calling these types of methods in the Handle Attributes and Methods Reference.

- You do not invoke a class constructor directly. To invoke a constructor, you pass constructor parameters in statements that participate in instantiating the class, such as the NEW function (classes), SUPER statement, or THIS-OBJECT statement. The constructor that is invoked matches the particular signature of the parameters that you pass.
- When you call the Publish( ) event method to publish a class event, or when you execute the PUBLISH statement to publish a named event, any parameters are passed to every event handler that is subscribed to the event. However, any parameter values returned from the Publish( ) method or PUBLISH statement reflect the settings of the last event handler to execute. Therefore, when you publish a class or named event, the value returned for an OUTPUT or INPUT-OUTPUT parameter, or for a member of any object (class or handle-based) referenced in an INPUT parameter, depends on the execution order of the event handlers subscribed to the event, which is not guaranteed. In addition, for INPUT-OUTPUT parameters or for members of objects referenced by INPUT parameters, the values returned from each event handler are used as input to the next event handler that executes.

  Caution: Because the execution order of multiple event handlers subscribed to a single event cannot be guaranteed, if you subscribe more than one event handler to a given class or named event, you cannot be certain which handler will return the values of any parameters that you define for the event.

- You cannot pass a CLOB or BLOB directly as a corresponding LONGCHAR/CHARACTER or MEMPTR parameter. You must first assign the CLOB to a LONGCHAR/CHARACTER or assign the BLOB to a MEMPTR and pass the result. For information on assigning large object data, see the Assignment (=) statement reference entry.
- When you instantiate a class dynamically, using the DYNAMIC-NEW statement, ABL distinguishes overloaded constructors only by the number of parameters.
- When you call overloaded methods or constructors that differ by a parameter that supports widening, ABL looks for an exact data type match at compile time. If an exact match is not found, the compiler will look for the closest widened match.
- When you pass an ABL array parameter, ABL initiates an array assignment that makes a deep copy of the array source to the array target, depending on the parameter mode. For more information on array assignments, see the Assignment (=) statement reference entry.
- For INPUT parameters of .NET methods or constructors defined with certain .NET data types, ABL supports widening relationships that allow you to pass ABL arguments with different ABL data types than those supported for implicit mapping to .NET data types (see Table 24: Implicit mappings between .NET and ABL data types in the Data types reference entry).

  The following table lists the .NET parameter data types for which ABL supports the widening of ABL data types passed as INPUT arguments. For each .NET parameter data type, the listed ABL implicit mapping data type represents the closest matching ABL data type that you can pass to a .NET input parameter defined with the matching .NET data type. The corresponding listed ABL INPUT widening data types can hold smaller values that might also be acceptable to the .NET input parameter. However, they might also adhere to similar limitations as defined for passing values as the listed ABL implicit mapping data type. (Again, see Table 24: Implicit mappings between .NET and ABL data types for a more complete description of these limitations.)

  Table 54: Data types for .NET INPUT parameter widening

.NET parameter object type	C# parameter primitive type	ABL implicit mapping data type	ABL INPUT widening data type
`System.DateTime`	N/A	DATETIME	DATE
`System.Decimal`	`decimal`	DECIMAL	INTEGER, INT64

.NET parameter object type	C# parameter primitive type	ABL implicit mapping data type	ABL INPUT widening data type
System.UInt32	uint	INT64	INTEGER[55]
System.Int64	long	INT64	INTEGER
System.UInt64	ulong	DECIMAL	INTEGER[55], INT64[55]
System.Double	double	DECIMAL	INTEGER, INT64
System.Single	float	DECIMAL	INTEGER[56], INT64[56]

- For OUTPUT parameters of .NET methods or constructors defined with certain .NET data types, ABL supports widening relationships that allow you to pass ABL arguments with different data types than those supported for implicit mapping to .NET data types (see Table 24: Implicit mappings between .NET and ABL data types in the Data types reference entry).

  The following table lists the .NET parameter data types for which ABL supports the widening of ABL data types passed as OUTPUT arguments. For each .NET parameter data type, the listed ABL implicit mapping data type represents the closest matching ABL data type that can hold a .NET value passed to the OUTPUT parameter defined with the matching .NET data type. The corresponding listed ABL OUTPUT widening data types might hold even larger values than the listed ABL implicit mapping data type. However, but they might also adhere to similar limitations as defined for the listed ABL implicit mapping data type (Again, see Table 24: Implicit mappings between .NET and ABL data types for a more complete description of these limitations.)

  Table 55: Data types for .NET OUTPUT parameter widening

.NET parameter object type	C# parameter primitive type	ABL implicit mapping data type	ABL OUTPUT widening data type
System.Byte	byte	INTEGER	INT64, DECIMAL
System.SByte	sbyte	INTEGER	INT64, DECIMAL
System.Char	char	CHARACTER	LONGCHAR
System.DateTime	N/A	DATETIME	DATETIME-TZ
System.Int16	short	INTEGER	INT64, DECIMAL
System.UInt16	ushort	INTEGER	INT64, DECIMAL
System.Int32	int	INTEGER	INT64, DECIMAL
System.UInt32	uint	INT64	DECIMAL
System.Int64	long	INT64	DECIMAL

[55] If you pass a negative ABL data type to an unsigned data type, the ABL virtual machine (AVM) raises a run-time error.
[56] You can lose precision if you pass an ABL INTEGER or INT64 to a System.Single parameter.

- If you are passing an ABL object reference to an OUTPUT or INPUT-OUTPUT parameter defined as an unmapped .NET type, ABL treats any type returned by .NET as a class.
- If you assign a .NET value type object returned as an OUTPUT parameter from a .NET method to an ABL object reference variable, the returned object referenced in ABL is a separate copy from the original value type instance maintained by .NET. However within the ABL session, the object reference to the ABL copy, when passed as a parameter, is passed by reference like any object reference to an ABL class instance.
- ABL allows you to pass ABL arrays (defined with EXTENT) and .NET array objects to each other. Passing an array parameter of any type constitutes an array assignment whose direction moves from the source to the target array according to the parameter mode. How this array assignment works for ABL and .NET array parameters depends upon the types of the source and target arrays. For more information, see the Data types reference entry.
- For temp-table parameters or temp-tables passed as part of a ProDataSet parameter, the AVM does not check for matching temp-table field data types if the SCHEMA-MARSHAL attribute for the temp-table is set to "NONE".
- You **cannot** pass a REFERENCE-ONLY buffer or a BEFORE-TABLE buffer as a BUFFER parameter. It also does not matter if a REFERENCE-ONLY buffer has been bound. Passing these kinds of buffers as a BUFFER parameter (as opposed to passing them as an INPUT or OUTPUT parameter) raises a compiler error. Instead of passing the buffer as a parameter, you **can** pass the buffer's temp-table as a BY-REFERENCE TABLE or TABLE-HANDLE parameter.
- You cannot pass a BUFFER parameter to a remote procedure. Otherwise, the AVM raises a run-time error.
- Class-based objects can be passed between an OpenEdge application server and an ABL client. The objects are serialized on the sending side, and a copy is deserialized on the receiving side. The following restrictions apply to objects being passed as parameters between an OpenEdge application server and client:
  - The object's class and all of the classes in its hierarchy must be marked as SERIALIZABLE. See the CLASS statement entry for more information.
  - All of the object's data members that are themselves defined as class-based objects must be of class types that are also marked SERIALIZABLE.
  - The class must be defined on both the client and server sides, and the data members, properties, methods signatures, and events on both sides must match exactly.
  - The content of static data members is not serialized, and the state of queries, buffers, open files, streams, and event subscriptions, for example, are not maintained.
  - Handle-based variables are serialized, but no information for reconstructing handle-based objects on the receiving side is serialized.
  - MEMPTRs assigned by an ABL application are serialized, but MEMPTRs from an external source (such as a DLL or shared library) are not serialized.
  - CHARACTER values are serialized using the codepage defined in `-cpinternal`, and the values are converted using the codepage defined for `-cpinternal` in the session that does the deserialization. LONGCHAR values are also converted using `-cpinternal` unless their codepage is fixed with the FIX-CODEPAGE statement, in which case the fixed codepage is used. An error is raised if the conversion fails.
  - Statically defined temp-tables and ProDataSets in user-defined classes are serialized, except for REFERENCE-ONLY tables.
  - The REJECTED, ERROR, ERROR-STRING, and DATA-SOURCE-MODIFIED attributes for temp-tables in an object are maintained as part of the deserialization process. Similarly, the REJECTED and ERROR attributes for ProDataSets are maintained during serialization.

- The only built-in object types that can be passed as parameters are `Progress.Json.ObjectModel.JsonObject`, `Progress.Json.ObjectModel.JsonArray`, and `Progress.Json.ObjectModel.ObjectModelParser`, and the built-in error objects.
- .NET and ABL-extended .NET objects cannot be passed as parameters.

For more information on passing class-based objects as parameters between an OpenEdge application server and an ABL client, see the *Progress Application Server for OpenEdge Application Migration and Development*.

- You can pass a temp-table or ProDataSet containing an object reference field as a parameter between an OpenEdge application server and an ABL client, as long as the objects themselves satisfy the restrictions placed on all class-based objects being passed between an OpenEdge application server and an ABL client.
- If you call a remote procedure asynchronously and pass a parameter as OUTPUT TABLE-HANDLE *`temp-table-handle`* APPEND, the event procedure or method must specify a corresponding DEFINE INPUT PARAMETER TABLE-HANDLE FOR *`temp-table-handle`* APPEND statement, and *`temp-table-handle`* must be global to both the calling procedure and the event procedure or method.
- If you pass a TABLE-HANDLE parameter to a method of a class or a class constructor, where the method or constructor is overloaded with corresponding TABLE-HANDLE and TABLE parameter definitions, the AVM executes the method or constructor that best matches the TABLE-HANDLE parameter. If the schema of the TABLE-HANDLE parameter matches the schema of the TABLE parameter definition, the AVM executes the method or constructor with the TABLE parameter definition. Otherwise, the AVM executes the method or constructor with the TABLE-HANDLE parameter definition.

  If you pass a TABLE parameter to a similarly overloaded method or constructor, the AVM executes the method or constructor with a matching TABLE parameter definition. If the AVM cannot identify a method or constructor with a matching TABLE parameter definition, it executes the method or constructor with the corresponding TABLE-HANDLE parameter definition.

  If the AVM cannot locate a method or constructor whose temp-table parameter definitions match the corresponding TABLE-HANDLE or TABLE parameter, the AVM raises a run-time error identifying the ambiguity.
- If you pass a DATASET-HANDLE parameter to a method of a class or a class constructor, where the method or constructor is overloaded with corresponding DATASET-HANDLE and DATASET parameter definitions, the AVM executes the method or constructor that best matches the DATASET-HANDLE parameter. If the schema of the DATASET-HANDLE parameter matches the schema of the DATASET parameter definition, the AVM executes the method or constructor with the DATASET parameter definition. Otherwise, the AVM executes the method or constructor with the DATASET-HANDLE parameter definition.

  If you pass a DATASET parameter to a similarly overloaded method or constructor, the AVM executes the method or constructor with a matching DATASET parameter definition. If the AVM cannot identify a method or constructor with a matching DATASET parameter definition, it executes the method or constructor with the corresponding DATASET-HANDLE parameter definition.

  If the AVM cannot locate a method or constructor whose ProDataSet parameter definitions match the corresponding DATASET-HANDLE or DATASET parameter, the AVM raises a run-time error identifying the ambiguity.
- If you pass a class instance to overloaded methods or constructors whose corresponding class parameters differ only within the same class hierarchy, ABL looks from the bottom of the class hierarchy, starting with the most derived class, and chooses any method or constructor whose corresponding parameters are defined for the same class as the passed parameter. Otherwise, the AVM chooses any method or constructor whose corresponding parameters are defined for the most derived class in the same class hierarchy, which is also a super class of the passed parameter.
- If you pass a class instance to overloaded methods or constructors whose corresponding class parameters differ by one or more interfaces that the class implements, only a method or constructor whose corresponding class parameter exactly matches the class of the passed parameter is acceptable to the compiler. Otherwise,

the AVM generates a compile-time error for ambiguity, as one implemented interface is no better match than another.

- If you pass the Unknown value (?) to a parameter of an overloaded method or constructor, the AVM only selects the correct method or constructor to execute when passing the Unknown value (?) causes no ambiguity among them. Otherwise, the AVM generates a compile-time error for ambiguity.
- If you pass an ABL expression to an overloaded method or constructor whose data type cannot be known at compile-time (for example, the BUFFER-VALUE attribute), the AVM can only call the correct method or constructor at run time, and only when the overloaded method or constructor differs by more than the parameter with an unknown data type. Otherwise, the AVM raises a run-time error for ambiguity.
- You can avoid compile-time ambiguity errors when passing the Unknown value (?) or an expression of an unknown data type in one of two ways:
  - Convert the Unknown value (?) or expression to a specific data type using the appropriate ABL data type conversion function to pass the value.
  - Assign the Unknown value (?) or expression to a variable of a specific data type to pass the value.
- If you pass a method of a class as a parameter to another method, constructor, procedure, or user-defined function that you invoke, and the method parameter executes the RETURN statement with the ERROR option, the method, constructor, procedure, or user-defined function that you invoke does not run. The AVM then raises ERROR on the invoked method, constructor, or procedure. The AVM does not raise ERROR on an invoked user-defined function.
- A parameter list that contains the word "DATASET" or "BUFFER", where the parameter is not a DATASET or BUFFER parameter, but an expression involving those keywords, may be ambiguous, causing a compiler error. If this happens, enclosing the expression in parentheses to identify the parameter as an expression will usually fix the ambiguity.

See also

Assignment (=) statement, Expression, FUNCTION statement, NEW function (classes), Publish( ) event method, PUBLISH statement, RUN statement, RUN SUPER statement, SUPER statement, SUPER system reference, THIS-OBJECT statement

PAUSE statement

Suspends processing indefinitely, or for a specified number of seconds, or until the user presses any key.

Note: Does not apply to SpeedScript programming.

Syntax

```
PAUSE
   [ n ]
   [ BEFORE-HIDE ]
   [ MESSAGE message | NO-MESSAGE ]
   [ IN WINDOW window ]
```

n

A numeric expression specifying the number of seconds that you want to suspend processing. You can choose a whole or a fractional value for the time-out interval. If the time-out period you specify is a fractional value, the value is rounded to the nearest whole millisecond.

If you do not use this option, the AVM suspends processing until the user presses any key.

BEFORE-HIDE

Specifies the pause action the user must take whenever frames are hidden automatically. If you specify *n*, *n* is the number of seconds the AVM pauses before hiding. If you do not specify *n*, the pause lasts until the user presses a key.

MESSAGE *message*

Displays the message "Press spacebar to continue" on the status line of the terminal screen when the AVM encounters a PAUSE statement. Use the MESSAGE option to override that default message. A *message* is a constant character string.

NO-MESSAGE

Tells the AVM to pause but not to display a message on the status line of the terminal screen.

IN WINDOW *window*

Specifies the window to which the pause action applies. The value *window* must be a handle to a window. If you do not use the IN WINDOW phrase, the PAUSE statement applies to the current window.

Example

The FOR EACH block in this procedure reads each of the records from the Customer table and displays information from each record. Because the DISPLAY uses a down frame (multiple records displayed in the frame), the AVM usually fills the window with as many records as possible and then displays the message: "Press spacebar to continue". The PAUSE 2 BEFORE-HIDE message tells the AVM to pause only two seconds before hiding the frame and displaying additional records.

r-pause.p

```
PAUSE 2 BEFORE-HIDE MESSAGE "Pausing 2 seconds".

FOR EACH Customer NO-LOCK WITH 13 DOWN:
  DISPLAY Customer.CustNum Customer.Name.
END.
```

Notes

- After you use PAUSE, that statement is in effect for all the procedures run in that session unless it is overridden by other PAUSE statements in those procedures, or until you return to the Editor.
- Using the PAUSE *n* BEFORE-HIDE statement is a good way to write a demonstration application that runs by itself.
- The AVM automatically pauses before removing a frame and displays the "Press spacebar to continue" message if you have not had a chance to see the data in the frame.

- When a PAUSE occurs, the AVM clears any keystrokes buffered from the keyboard, discarding any type-ahead characters.

PDBNAME function

Returns the physical name of a currently connected database.

Syntax

```
PDBNAME ( integer-expression | logical-name | alias )
```

integer-expression

If the parameter supplied to PDBNAME is an integer expression, and there are, for example, three currently connected databases, then PDBNAME(1), PDBNAME(2), and PDBNAME(3) return their physical names. Also, continuing the same example of three connected databases, PDBNAME(4), PDBNAME(5), etc., return the Unknown value (?).

logical-name | *alias*

This form of the PDBNAME function requires a quoted character string or a character expression as a parameter. If the parameter is the logical name of a connected database or an alias of a connected database, then the physical name is returned. Otherwise, it returns the Unknown value (?).

Example

This procedure finds the physical name of the database that currently has the DICTDB alias:

r-pdbnam.p

```
MESSAGE "The current DICTDB is" PDBNAME("DICTDB") + ".db".
```

Notes

The old DBNAME function has been retained for compatibility and is equivalent to PDBNAME(1).

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, SDBNAME function

PRESELECT phrase

Specifies a set of records to preselect for a DO or REPEAT block.

Syntax

```
PRESELECT
  [ EACH | FIRST | LAST ]record-phrase
  [ , [ EACH | FIRST | LAST ] record-phrase ]...
  [[ BREAK ]
    { BY expression[ DESCENDING ]
      | COLLATE ( string , strength[ , collation] ) [ DESCENDING ]
    }...]
```

[EACH | FIRST | LAST]*record-phrase*

Goes through a table, selecting records that meet the criteria you specify in *record-phrase*. PRESELECT creates a temporary index that contains pointers to each of the preselected records in the database table. Then you can use other statements, such as FIND NEXT, within the block to process those records.

The *record-phrase* option identifies the criteria to use when preselecting records. Following is the syntax for the *record-phrase*:

```
{record [field-list]}
[constant]
[[ LEFT ] OUTER-JOIN ]
[ OF table ]
[ WHERE expression ]
[ TENANT-WHERE expression [ SKIP-GROUP-DUPLICATES ]]
[ USE-INDEX index ]
[ USING [ FRAME frame ]field
  [ AND [ FRAME frame ]field]...
 ]
[ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ]
[ NO-PREFETCH ]
```

Specifying multiple occurrences of *record-phrase* preselects the tables using an inner join. Also, any sorting you specify applies to all the tables. If you then do a FIND on the last table in the PRESELECT list, the AVM reads records into the buffers for all of the tables in the list.

For more information on *record-phrase* and inner joins, see the Record phrase reference entry.

BREAK

When used in combination with the FIRST function, LAST function, FIRST-OF function, and LAST-OF function, BREAK indicates that subgroups are used for aggregation. If you use BREAK, you must also use BY.

```
BY expression [ DESCENDING ]
```

Sorts the preselected records by the value of *expression*. If you do not use the BY option, PRESELECT sorts the records in order by the index used to extract the records. The DESCENDING option sorts the records in descending order (not in the default ascending order).

```
COLLATE ( string , strength [ , collation ] ) [ DESCENDING ]
```

Generates the collation value of a string after applying a particular strength, and optionally, a particular collation. The DESCENDING option sorts the records in descending order (not in default ascending order).

string

A CHARACTER expression that evaluates to the string whose collation value you want to generate.

strength

A CHARACTER expression that evaluates to an ABL comparison strength or an International Components for Unicode (ICU) comparison strength.

The ABL comparison strengths include:

- **RAW** — Generates a collation value for the string based on its binary value.
- **CASE-SENSITIVE** — Generates a case-sensitive collation value for the string based on a particular collation. If you specify this strength with an ICU collation, the AVM applies the ICU TERTIARY strength.
- **CASE-INSENSITIVE** — Generates a case-insensitive collation value for the string based on a particular collation. If you specify this strength with an ICU collation, the AVM applies the ICU SECONDARY strength.
- **CAPS** — Generates a collation value for the string based on its binary value after converting any lowercase letters in the string to uppercase letters, based on the settings of the Internal Code Page (`-cpinternal`) and Case Table (`-cpcase`) startup parameters.

The ICU comparison strengths include:

- **PRIMARY** — Generates a collation value for the base characters in the string.
- **SECONDARY** — Generates a collation value for the base characters and any diacritical marks in the string.
- **TERTIARY** — Generates a case-sensitive collation value for the base characters and any diacritical marks in the string.
- **QUATERNARY** — Generates a case-sensitive collation value for the base characters and any diacritical marks in the string, and distinguishes words with and without punctuation. ICU uses this strength to distinguish between Hiragana and Katakana when applied with the ICU-JA (Japanese) collation. Otherwise, it is the same as TERTIARY.

Note: Use ICU comparison strengths only with ICU collations.

collation

A CHARACTER expression that evaluates to the name of an ABL collation table or ICU collation. If *collation* does not appear, COLLATE uses the collation table of the client.

The AVM reports an error and stops execution if one of the following occurs:

- *strength* does not evaluate to a valid value.
- *collation* does not evaluate to a collation table residing in the convmap.cp file.
- *collation* evaluates to a collation table that is not defined for the code page corresponding to the -cpinternal startup parameter.

Examples

To process a multi-table collection gathered by the PRESELECT option, use the last table named in the collection when you want to read the selected records. The AVM then automatically retrieves records from the other tables.

r-presl1.p

```
REPEAT PRESELECT EACH Order, Customer OF Order, EACH OrderLine OF Order
  BY Order.OrderDate BY Order.CustNum BY OrderLine.ItemNum:
  FIND NEXT OrderLine.
  DISPLAY Order.OrderDate Order.CustNum Customer.Name OrderLine.ItemNum.
END.
```

The PRESELECT option in this example selects the logically joined record that consists of Order, OrderLine, and Customer, and makes all of these records available in the REPEAT block. Usually you perform more complex processing within the PRESELECT block.

If, within a PRESELECT block, you find a record using the ROWID of that record, the AVM disregards any other selection criteria you applied to the PRESELECT. For example, suppose the ROWID of Order number 4 is stored in the variable ord-rowid:

```
DO PRESELECT EACH Order NO-LOCK WHERE Order.OrderNum > 5:
  FIND FIRST Order NO-LOCK WHERE ROWID(Order) = ord-rowid.
  DISPLAY Order.
END.
```

In this example, the AVM finds and displays order number 4 even though the selection criteria specifies that the order number must be greater than 5. The ROWID always overrides other selection criteria. Furthermore, if you use FIND...WHERE ROWID(*record*) =..., the index cursor is not reset in the preselected list. That is, even if record ROWID(*record*) is in the preselected list, FIND NEXT does **not** find the record that follows it in the preselected list.

See also

DEFINE BUFFER statement, DO statement, FIND statement, REPEAT statement

PROC-HANDLE function

Returns a value in the appropriate data type (usually INTEGER) that is a unique identifier for a stored procedure.

Syntax

```
PROC-HANDLE
```

Example

This procedure runs the stored procedure pcust and writes the procedure handle to the variable iHandle. It writes the results of the stored procedure identified by this procedure handle into the ABL-supplied buffer, proc-text-buffer, and displays it.

```
DEFINE VARIABLE iHandle AS INTEGER NO-UNDO.

RUN STORED-PROCEDURE pcust iHandle = PROC-HANDLE (10, OUTPUT 0, OUTPUT 0).
FOR EACH proc-text-buffer WHERE PROC-HANDLE = iHandle:
  DISPLAY proc-text.
END.
CLOSE STORED-PROCEDURE pcust WHERE PROC-HANDLE = iHandle.
```

Notes

- Progress Software Corporation recommends that you specify a procedure handle for each stored procedure that you run.
- You do not have to specify a handle if there is only one active stored procedure and you do not include SQL statements in the ABL application. In the case of ORACLE only, the DataServer passes SQL statements to the ORACLE RDBMS and uses the default system handle in the process.

For more information on using this function, see the OpenEdge DataServer Guides (*Use the Microsoft SQL Data Server* and *Use the Oracle Data Server*).

See also

CLOSE STORED-PROCEDURE statement, PROC-STATUS function, RUN STORED-PROCEDURE statement

PROC-STATUS function

Returns the return status from a stored procedure. The return status is an INTEGER value that indicates whether a stored procedure failed and why.

Syntax

```
PROC-STATUS
```

Example

This procedure runs the ORACLE stored procedure pcust and writes the results of the stored procedure into the ABL-supplied buffer, proc-text-buffer. The CLOSE STORED-PROCEDURE statement then retrieves the output parameters. The return status is written to the variable `iStat` and is displayed. This same code works for accessing a stored procedure from an Microsoft SQL Server-compliant data source:

```
DEFINE VARIABLE iStat AS INTEGER NO-UNDO.

RUN STORED-PROCEDURE pcust (10, OUTPUT 0, OUTPUT 0).
FOR EACH proc-text-buffer:
END.
CLOSE STORED-PROCEDURE pcust iStat = PROC-STATUS.
DISPLAY iStat.
```

Notes

- For descriptions of the possible values for the return status of a non-ABL stored procedure, see the procedure's documentation.
- For more information on using this function, see the OpenEdge DataServer Guides (*Use the Microsoft SQL Data Server* and *Use the Oracle Data Server*).

See also

CLOSE STORED-PROCEDURE statement, PROC-HANDLE function, RUN STORED-PROCEDURE statement

PROCEDURE statement

Defines an internal procedure as an ABL procedure or declares an internal procedure prototype for an external routine in a Windows dynamic link library (DLL) or UNIX shared library, or for an internal ABL procedure defined in an external procedure that is itself a super procedure of the declaration procedure. The following syntax boxes describe the syntax for each use of the statement, beginning with an internal procedure definition.

Syntax

```
PROCEDURE proc-name [ PRIVATE ] :
  [procedure-body]
```

This is the syntax to declare an internal procedure prototype for a routine in a Windows DLL or UNIX shared library, or for an internal ABL procedure defined in a super procedure:

```
PROCEDURE proc-name
  {   EXTERNAL "dllname" [ CDECL | PASCAL | STDCALL ]
        [ ORDINAL n ] [ PERSISTENT ] [ THREAD-SAFE ]
    | IN SUPER } :
  [ procedure-body ]
```

proc-name

The name of the internal procedure.

To define the name of an internal ABL procedure that is an event handler for ActiveX controls (OCX event procedure), you must specify *proc-name* according to the following syntax:

```
{control-frame-name.control-name.event-name | ANYWHERE.event-name}
```

For more information on naming event handlers for ActiveX controls using this syntax, see the notes for this reference entry.

EXTERNAL "*dllname*"

Declares the internal procedure as a Windows DLL or UNIX shared library routine. The *dllname* argument, specified as a string literal, is the name of the DLL or library containing the routine. The value of *dllname* can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

CDECL

Tells ABL to use the C calling convention when accessing the routine.

PASCAL

Supported only for backward compatibility. This option is not valid for SpeedScript.

STDCALL

Tells ABL to use the standard Windows calling convention when accessing the routine. This is the default.

Note: The 64-bit Windows GUI and character clients ignore the CDECL, PASCAL, and STDCALL calling conventions if they are specified. The 64-bit Windows GUI and character clients always use the standard 64-bit FASTCALL calling convention.

ORDINAL *n*

Specifies the number of the DLL entry point (the *n*th routine) to invoke. If you use the ORDINAL option, then *proc-name* can specify any name used in the corresponding RUN statement to reference the routine. If you omit the ORDINAL option, *proc-name* specifies which DLL routine you want to invoke.

For UNIX shared library routines, this option does not apply and is ignored.

PERSISTENT

Specifies that the DLL or shared library routine should remain loaded in memory until the AVM exits or the session executes the RELEASE EXTERNAL statement.

THREAD-SAFE

Specifies that the DLL or shared library routine is thread safe. When a DLL or shared library is marked as THREAD-SAFE, multiple sessions can access it simultaneously. This option is only valid when

running an application on an instance of the Progress Application Server for OpenEdge; otherwise, this option does not apply and is ignored.

PRIVATE

Indicates the following about the internal procedure:

- It cannot be invoked from an external procedure—that is, from a procedure file external to the current procedure file.
- The INTERNAL-ENTRIES attribute on the procedure that defines it does not provide its name (unless the procedure that defines it is the current procedure file).
- The GET-SIGNATURE method on the procedure that defines it does not provide its signature (unless the procedure that defines it is the current procedure file).

IN SUPER

Declares that the definition of the internal procedure resides in a super procedure.

procedure-body

The body of an internal procedure definition. Define *procedure-body* using the following syntax:

```
procedure-logic
      .
      .
      .
   [ catch-block [ catch-block... ]]
   [ finally-block ]
[ END [ PROCEDURE ] . ]
```

procedure-logic

Zero or more ABL statements, depending on the internal procedure definition or declaration. Each logic statement must end in with a period (.).

If you declare the internal procedure as an ABL procedure, these statements can include executable statements and non-executable statements including definitions of run-time parameters (using the DEFINE PARAMETER statement), local program variables, frames, widgets, and buffers. Any such objects you define within the internal procedure remain in effect only for the life of the internal procedure.

If you are defining the internal procedure for use as an event procedure to handle asynchronous remote requests, you can specify run-time parameters as INPUT only. (Any other type of parameter generates a run-time error.) Each INPUT parameter must correspond in order and data type with an OUTPUT (or INPUT-OUTPUT) parameter as defined in the remote procedure that executes the request.

If you declare the internal procedure as a DLL or UNIX shared library routine (using the EXTERNAL option), these statements can include only DEFINE PARAMETER statements.

For more information on accessing DLL or UNIX shared library routines from ABL, see the topics on DLLs in *OpenEdge Programming Interfaces*.

catch-block

Specifies a CATCH statement that defines error handling code for one or more error types. A DO block does not have any default error handling. Therefore, a DO block must have error handling options specified such that it becomes an undoable block. Otherwise, ABL generates a compiler warning. For more information on *catch-block*, see the CATCH statement reference entry.

finally-block

Specifies a FINALLY statement that defines the processing that must occur after all other processing in the block occurs. For more information on *finally-block*, see the FINALLY statement reference entry.

END [PROCEDURE]

Specifies the end of the internal procedure body. If *procedure-logic* contains one or more statements, you must end the internal procedure body with the END statement.

Example

The following example declares an ABL internal procedure that computes the factorial of an integer entered as an INPUT parameter. The result is returned as an OUTPUT parameter. Note that the following procedure calls itself recursively to obtain the result:

r-factrl.p

```
DEFINE VARIABLE FactorialResult AS INTEGER NO-UNDO FORMAT ">>>,>>>,>>9".
DEFINE VARIABLE FactorialInput  AS INTEGER NO-UNDO.

REPEAT:
  SET FactorialInput VALIDATE(FactorialInput <= 12 AND FactorialInput >= 0,
    "Value must be between 0 and 12.").
  RUN Factorial (INPUT FactorialInput, OUTPUT FactorialResult).
  DISPLAY FactorialResult.
END.

PROCEDURE Factorial:
  DEFINE INPUT PARAMETER  PTerm           AS INTEGER NO-UNDO.
  DEFINE OUTPUT PARAMETER FactorialResult AS INTEGER NO-UNDO.

  DEFINE VARIABLE WorkingResult AS INTEGER NO-UNDO.

  IF PTerm <= 1 THEN DO:
    FactorialResult = 1.
    RETURN.
  END.
  ELSE DO:
    RUN Factorial (INPUT PTerm - 1, OUTPUT WorkingResult).
    FactorialResult = PTerm * WorkingResult.
  END.
END PROCEDURE.
```

The following example declares a DLL routine, `MessageBox()`, which displays a message:

r-dllex1.p

```
DEFINE VARIABLE iResult AS INTEGER NO-UNDO.

MESSAGE "  It's a whole new world!"
  VIEW-AS ALERT-BOX MESSAGE BUTTONS OK TITLE "ABL Message".

RUN MessageBoxA (0, "  It's a whole new world, again!!",
  "ABL DLL Access", 0, OUTPUT iResult).

PROCEDURE MessageBoxA EXTERNAL "user32.dll":
  DEFINE INPUT PARAMETER hwnd    AS LONG.
  DEFINE INPUT PARAMETER mbtext  AS CHARACTER.
  DEFINE INPUT PARAMETER mbtitle AS CHARACTER.
  DEFINE INPUT PARAMETER style   AS LONG.
  DEFINE RETURN PARAMETER result AS LONG.
END.
```

The following code fragment declares a UNIX shared library routine:

```
PROCEDURE atoi EXTERNAL "/usr/lib/libc.so.1":
...
```

Notes

- You can terminate a PROCEDURE statement with either a period (.) or a colon (:), but typically use a colon (:) for a procedure definition or prototype that includes *procedure-body* and a period (.) for a procedure definition or prototype that omits any *procedure-body*.
- You can place an internal procedure definition or declaration before, after, or in the middle of your main procedure code. You **cannot** nest an internal procedure within another internal procedure.
- Use the RUN statement to invoke an internal procedure. You can run an internal procedure from within the external procedure that defines it, either from the main-line of the external procedure or from another internal procedure defined in the external procedure. You can also run an internal procedure defined in another external procedure using the IN *proc-handle* option of the RUN statement as long as the external procedure meets one of these conditions:
  - It is active on the procedure call stack
  - It is an instance of a persistent procedure
- You cannot define shared objects, work tables, or temp-tables within an internal procedure.
- An internal procedure can reference any objects defined in the outer procedure block. For example, it can reference variables, buffers (explicit or implicit; shared or unshared), variables, run-time parameters, named frames, or temp-tables. If you define an object with the same name in the internal procedure and the external procedure, a reference within the internal procedure resolves to the local object.
- A buffer explicitly defined in an internal procedure is scoped to the internal procedure. Any other buffers are scoped to the outer procedure block.

- To define the internal procedure as an event handler for ActiveX controls (OCX event procedure), you must specify *proc-name* according to the following syntax:

```
{ control-frame-name .control-name .event-name
  | ANYWHERE .event-name
}
```

  In *control-frame-name*.*control-name*.*event-name*, *control-frame-name* is the name (unquoted) of the control-frame that contains the ActiveX control. This is the name that the AppBuilder typically assigns to the control-frame (NAME widget attribute) when you insert the control into your user interface. The *control-name* is the value (unquoted) that you assign to the control Name property at design time in the AppBuilder Property Window. The *event-name* is the name (unquoted) of the ActiveX control event that you want to trigger execution of this procedure.

  In ANYWHERE.*event-name*, ANYWHERE specifies an event procedure that handles the specified event in any ActiveX control. This event procedure executes only if you have not defined a *control-frame-name*.*control-name*.*event-name* event procedure that exactly matches the control/event combination at run time.

  At design time, the AppBuilder lists the available events for a control and automatically creates a template for the OCX event procedure definition from the event that you select. For more information on how to create OCX event procedures in the AppBuilder, see the information on ActiveX controls in *OpenEdge Programming Interfaces*. For more information on how to work with OCX event procedures in an application, see *OpenEdge Programming Interfaces*.

- When you define an OCX event procedure, you can access the component handle (COM-HANDLE value) of the control that generates the event at run time using the COM-SELF system handle. You can also access the handle of the parent control-frame using the SELF system handle.
- The RETURN-VALUE function provides the value returned by the most recently executed RETURN statement of a local or remote procedure.
- You use the call object handle to dynamically invoke a Windows DLL routine or UNIX shared library routine at run time.

See also

Call object handle, COM-SELF system handle, DEFINE PARAMETER statement, END statement, RUN statement, TRIGGER PROCEDURE statement

PROCESS-ARCHITECTURE function

Identifies the bit value of the AVM process you are using at run time and returns this value as an INTEGER. Valid values are 32 for a 32-bit AVM and 64 for 64-bit AVM. For example, an application that uses a 32-bit Windows API through the DLL external procedure interface may need to pass different parameters to a function when running a 64-bit GUI client than when running a 32-bit GUI client. Using PROCESS-ARCHITECTURE, you can decide the parameters to pass.

Syntax

```
PROCESS-ARCHITECTURE
```

Example

This procedure displays a message indicating whether the AVM is a 32-bit or 64-bit process, whether the process is running on a 32-bit or 64-bit Windows machine:

r-prbits.p

```
DEFINE VARIABLE hProcess AS INTEGER NO-UNDO.
DEFINE VARIABLE Wow64Process AS INTEGER NO-UNDO.
DEFINE VARIABLE resultValue AS INTEGER NO-UNDO.

IF PROCESS-ARCHITECTURE = 64 THEN
  /* 64-bit processes always run on 64-bit Windows */
  MESSAGE "64-bit process running on 64-bit Windows".
ELSE DO:
  /* If a 32-bit process is running under WOW64 it must be
  ** running on 64-bit Windows. Otherwise, it is running on
  ** 32-bit Windows.
  */
  RUN GetCurrentProcess(OUTPUT hProcess).
  RUN IsWow64Process(hProcess, INPUT-OUTPUT Wow64Process, OUTPUT resultValue).
  MESSAGE "32-bit process running on " + (IF Wow64Process > 0 THEN "64" ELSE "32") +
"-bit Windows".
END.

PROCEDURE GetCurrentProcess EXTERNAL "kernel32.dll":
  DEFINE RETURN PARAMETER hProcess AS LONG.
END.

PROCEDURE IsWow64Process EXTERNAL "kernel32.dll":
  DEFINE INPUT PARAMETER hProcess AS LONG.
  DEFINE INPUT-OUTPUT PARAMETER Wow64Process AS HANDLE TO LONG.
  DEFINE RETURN PARAMETER resultValue AS LONG.
END.
```

Notes

This function has a corresponding preprocessor. {&PROCESS-ARCHITECTURE} expands to either "32" or "64".

See also

{ } Preprocessor name reference

PROCESS EVENTS statement

Processes all outstanding events without blocking for user input.

Syntax

```
PROCESS EVENTS
```

Example

This procedure counts to 1,000 until you choose **STOP**:

r-proevs.p

```
DEFINE VARIABLE ix       AS INTEGER NO-UNDO.
DEFINE VARIABLE stop-sel AS LOGICAL NO-UNDO.

DEFINE BUTTON stop-it LABEL "STOP".

DISPLAY stop-it.

ON CHOOSE OF stop-it
  stop-sel = TRUE.

ENABLE stop-it.

DO ix = 1 TO 1000:
  DISPLAY ix VIEW-AS TEXT.
  PROCESS EVENTS.
  IF stop-sel THEN LEAVE.
END.
```

On each pass through the loop, the procedure displays the new value of `ix` and then checks whether any events are waiting to be processed. If no events have occurred, execution continues and the loop iterates. If the STOP button has been chosen, that event is processed changing the value of stop-sel. When execution continues, the program exits the loop.

If the loop does not contain the PROCESS EVENTS statement, the choose event never processes and the loop iterates until ix equals 1,000.

Notes

- The WAIT-FOR statement processes all pending events and blocks all other execution until a specified event occurs. The PROCESS EVENTS statement processes all pending events and immediately continues execution with the next statement.
- If there are any asynchronous requests for which PROCEDURE-COMPLETE events have been received but not yet processed, this statement processes these events as described for the WAIT-FOR statement.
- You cannot call the .NET method system.Windows.Forms.Application:DoEvent( ) in ABL. The PROCESS EVENTS statement performs the function of this method.
- .NET can raise exceptions in the context of an ABL session when this statement executes.
- In the context of the .NET blocking method, `System.Windows.Forms.Application:Run( )`, if you execute the PROCESS EVENTS statement for a non-modal ABL window from within a user-defined function or non-VOID method, this statement raises the STOP condition. For more information on the .NET `Application:Run( )` method, see the reference entry for the WAIT-FOR statement (.NET and ABL).

See also

WAIT-FOR statement (ABL only), WAIT-FOR statement (.NET and ABL)

PROGRAM-NAME function

Returns the name of the calling program.

Syntax

```
PROGRAM-NAME( n )
```

n

The numeric argument. If *n* is 1, the name of the current program is returned. If *n* is 2, the name of the calling program is returned. If there is no calling program then you have reached the top of the call stack and the AVM returns the Unknown value (?).

Example

This procedure returns the names of any procedure(s) that called it and displays the number of levels that the procedure was nested:

r-prgnm.p

```
/* Note this program should be run as a subroutine. */
/* The deeper the nesting, the better the illustration. */

DEFINE VARIABLE level AS INTEGER NO-UNDO INITIAL 1.

REPEAT WHILE PROGRAM-NAME(level) <> ?:
  DISPLAY LEVEL PROGRAM-NAME(level) FORMAT "x(30)".
  level = level + 1.
END.
```

Notes

- If you execute a procedure directly from the Procedure Editor or the AppBuilder, then PROGRAM-NAME(1) returns the name of a temporary file rather than the name of the actual procedure file.
- The PROGRAM-NAME function is useful when developing on-line help. For example, you can use the following code in your help routine to produce a program trace:

  r-trace.p

```
DEFINE VARIABLE ix    AS INTEGER   NO-UNDO INITIAL 2.
DEFINE VARIABLE plist AS CHARACTER NO-UNDO FORMAT "x(70)".

FORM plist
  WITH FRAME what-prog OVERLAY ROW 10 CENTERED 5 DOWN NO-LABELS
  TITLE " Program Trace ".

/* ix = 2, so skip the current routine: PROGRAM-NAME(1) */
DO WHILE PROGRAM-NAME(ix) <> ?:
  IF ix = 2 THEN
    plist = "Currently in       : " + PROGRAM-NAME(ix).
  ELSE
    plist = "Which was called by: " + PROGRAM-NAME(ix).

  ix = ix + 1.
  DISPLAY plist WITH FRAME what-prog.
  DOWN WITH FRAME what-prog.
END.

PAUSE.
HIDE FRAME what-prog.
```

- If the procedure you reference is an internal procedure, then PROGRAM-NAME returns a string with the following form:

```
"internal-procedure-name source-file-name"
```

- If the procedure you reference is a user interface trigger associated with a widget, then PROGRAM-NAME returns a string with the following form:

```
"USER-INTERFACE-TRIGGER source-file-name"
```

- If the procedure you reference is a user interface trigger that uses the ANYWHERE keyword, then PROGRAM-NAME returns a string with the following form:

```
"SYSTEM-TRIGGER source-file-name"
```

- If the procedure you reference is a session database trigger, then PROGRAM-NAME returns a string with the following form:

```
"type-TRIGGER source-file-name"
```

  Where *type* is either ASSIGN, CREATE, DELETE, FIND, or WRITE.

- If the call stack contains a method reference, then PROGRAM-NAME returns a string with the following form:

```
"method-name class-file-name"
```

  Where *class-file-name* is the name of the class definition (`.cls`) file in which *method-name* is implemented.

PROGRESS function

Returns one of the following character values which identifies the ABL product that is running: Full, Query or Run-time. Can also return COMPILE if you use the Developer's Toolkit, or COMPILE-ENCRYPT if you use the run-time Compiler.

Note: Does not apply to SpeedScript programming.

Syntax

```
PROGRESS
```

Examples

The following procedure uses the PROGRESS phrase function to determine which exit prompt is displayed on a menu:

r-progfn.p

```
/* Depending on the version of PROGRESS you are running, the main menu reflects
   available features for end-user */

DEFINE VARIABLE menu        AS CHARACTER NO-UNDO EXTENT 3.
DEFINE VARIABLE exit-prompt AS CHARACTER NO-UNDO.

IF PROGRESS EQ "FULL" THEN
  exit-prompt = " 3. Return to Full Editor ".
ELSE IF PROGRESS EQ "QUERY" THEN
  exit-prompt = " 3. Return to Query Editor".
ELSE IF PROGRESS EQ "RUN-TIME" THEN
  exit-prompt = "3. Exit Program".

DO WHILE TRUE:
  DISPLAY
    "  1. Display Customer Data" @ menu[1] SKIP
    "  2. Display Order Data"    @ menu[2] SKIP
    exit-prompt                  @ menu[3]
    FORMAT "x(26)" SKIP
  WITH FRAME choices NO-LABELS.

  CHOOSE FIELD menu AUTO-RETURN WITH FRAME choices
    TITLE "Demonstration menu" CENTERED ROW 10.
  HIDE FRAME choices.

  IF FRAME-INDEX EQ 1 THEN MESSAGE
    "You picked option 1.".
  ELSE IF FRAME-INDEX EQ 2 THEN MESSAGE
    "You picked option 2.".
  ELSE IF FRAME-INDEX EQ 3 THEN RETURN.
END.
```

This procedure displays a message that tells you the type of ABL product you are using:

```
MESSAGE "You are currently running this PROGRESS product:" PROGRESS
  VIEW-AS ALERT-BOX INFORMATION BUTTONS OK.
```

See also

DBVERSION function, PROVERSION function

PROMPT-FOR statement

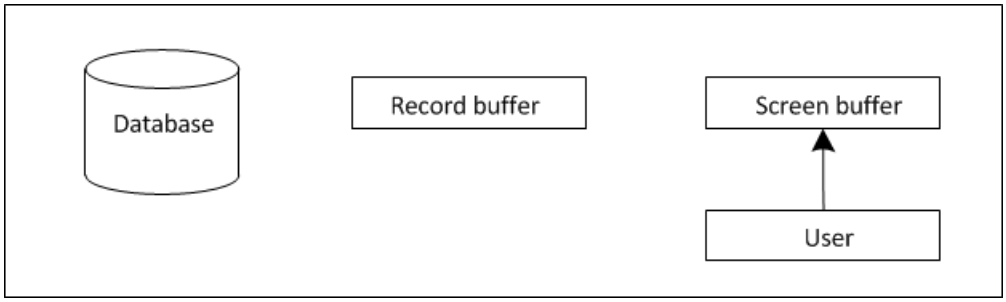
Requests input and places that input in the screen buffer (frame).

The PROMPT-FOR statement is a combination of the following statements:

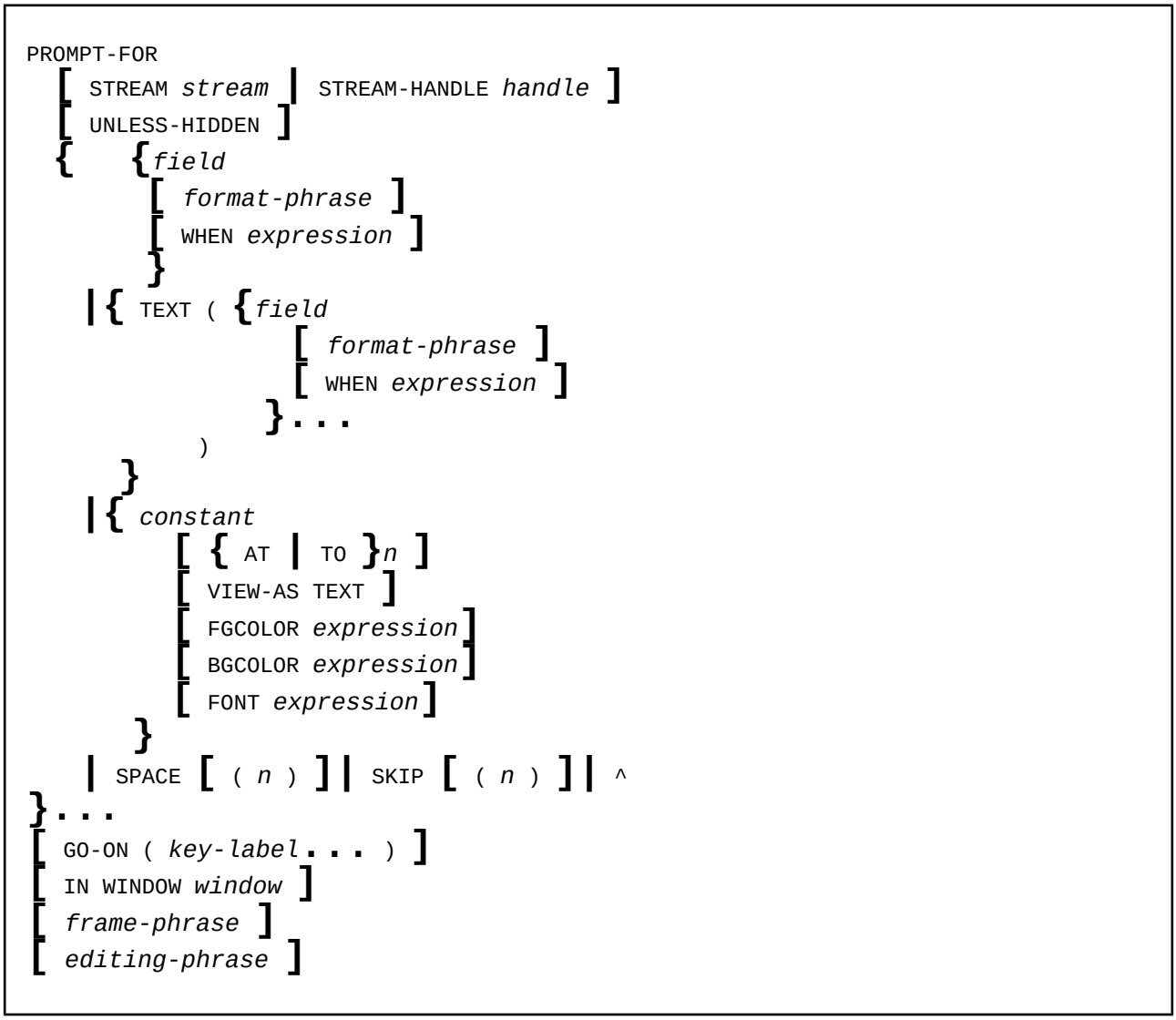
- **ENABLE** — Enables the specified field-level widgets (in this case fill-in fields) for input
- **WAIT-FOR** — Blocks for input and processes all ABL events until a specific ABL event occurs, in this case the GO universal key function event
- **DISABLE** — Disables the specified field-level widgets (in this case fill-in fields) for input

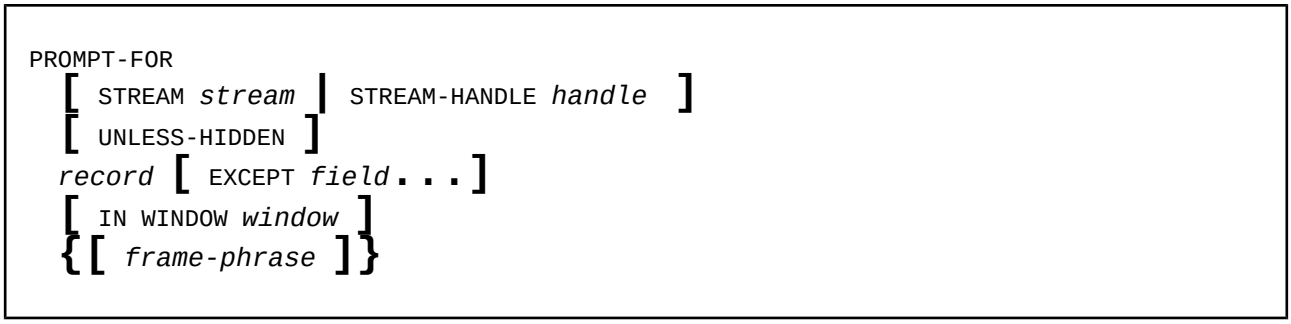
Note: Does not apply to SpeedScript programming.

Data movement



Syntax





STREAM *stream*

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream. See the DEFINE STREAM statement reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams.

STREAM-HANDLE *handle*

Specifies the handle to a stream. If *handle* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

UNLESS-HIDDEN

Restricts PROMPT-FOR to fields whose HIDDEN attribute is FALSE.

field

Specifies the name of the field or variable whose value you want to enter and store in the screen buffer. Remember that the PROMPT-FOR statement only accepts input and stores it in the screen buffer. The underlying record buffer of a field or variable is unaffected.

This *field* parameter is demonstrated in the following program:

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO INITIAL 3.

PROMPT-FOR ix.
MESSAGE "Record buffer" ix SKIP(0) "Screen buffer" INPUT x.
```

The program does the following:

- Stores the initial value of ix in a record buffer
- Prompts for a new value of ix, and stores the new value in a screen buffer
- Displays the value in the record buffer, retrieves the value in the screen buffer, then displays that

In the case of array fields, array elements with constant subscripts are treated just like any other field. Array fields with no subscripts or in the FORM statement are expanded as though you had typed in the implicit elements. See the DISPLAY statement reference entry for information on how array fields with expressions as subscripts are handled.

format-phrase

Specifies one or more frame attributes for a field, variable, or expression. For more information on *format-phrase*, see the Format phrase reference entry.

WHEN *expression*

Prompts for the field only when *expression* has a value of TRUE. Here, *expression* is a field name, variable name, or expression that evaluates to a LOGICAL value.

TEXT

Defines a group of character fields or variables (including array elements) to use automatic word-wrap. The TEXT option works only with character fields. When you insert data in the middle of a TEXT field, the AVM wraps data that follows into the next TEXT field, if necessary. If you delete data from the middle of a TEXT field, the AVM wraps data that follows into the empty area.

If you enter more characters than the format for the field allows, the AVM discards the extra characters. The character fields must have formats of the form x(n). A blank in the first column of a line marks the beginning of a paragraph. Lines within a paragraph are treated as a group and will not wrap into other paragraphs.

The following table lists the keys you can use within a TEXT field and their actions.

Table 56: Key actions in a TEXT field

Key	Action
APPEND-LINE	Combines the line the cursor is on with the next line.
BACK-TAB	Moves the cursor to the previous TEXT field.
BREAK-LINE	Breaks the current line into two lines beginning with the character the cursor is on.
BACKSPACE	Moves the cursor one position to the left and deletes the character at that position. If the cursor is at the beginning of a line, **BACKSPACE** moves the cursor to the end of the previous line.
CLEAR	Clears the current field and all fields in the TEXT group that follow.
DELETE-LINE	Deletes the line the cursor is on.
NEW-LINE	Inserts a blank line below the line the cursor is on.
RECALL	Clears fields in the TEXT group and returns initial data values for the group.
RETURN	If you are in overstrike mode, moves to the next field in the TEXT group on the screen. If you are in insert mode, the line breaks at the cursor and the cursor is positioned at the beginning of the new line.
TAB	Moves to the field after the TEXT group on the screen.If there is no other field, the cursor moves to the beginning of the TEXT group.

In this procedure, the `s-com`, or Order `Comments` field is a TEXT field. Run the following procedure and enter text in the field to see how the TEXT option works:

r-text.p

```
DEFINE VARIABLE s-com AS CHARACTER NO-UNDO FORMAT "x(40)" EXTENT 5.

FORM
  "Shipped   :" Order.ShipDate AT 13 SKIP
  "Misc Info :" Order.Instructions AT 13 SKIP(1)
  "Order Comments :" s-com AT 1
  WITH FRAME o-com CENTERED NO-LABELS TITLE "Shipping Information".

FOR EACH Customer NO-LOCK, EACH Order OF Customer:
  DISPLAY Customer.CustNum Customer.Name Order.OrderNum Order.OrderDate
    Order.PromiseDate WITH FRAME order-hdr CENTERED.
  UPDATE Order.ShipDate Order.Instructions TEXT(s-com)
    WITH FRAME o-com.
  s-com = "".
END.
```

constant [AT n | TO *n*] [VIEW-AS TEXT] [FGCOLOR *expression*]

[BGCOLOR *expression*] [FONT *expression*]

Specifies a literal value that you want displayed in the frame. If you use the AT option, *n* is the column in which you want to start the display. If you use the TO option, *n* is the column in which you want to end the display. You can use the BGCOLOR, FGCOLOR, and FONT options to define the colors and font in which the constant is displayed. If you use the VIEW-AS TEXT option, the constant is displayed as a text widget rather than a fill-in field.

SPACE [(*n*)]

Identifies the number (*n*) of blank spaces to insert after the field is displayed. The *n* can be 0. If the number of spaces you specify is more than the spaces left on the current line of the frame, a new line is started and any extra spaces are discarded. If you do not use this option or *n*, one space is inserted between items in the frame.

SKIP [(*n*)]

Identifies the number (*n*) of blank lines to insert after the field is displayed. The *n* can be 0. If you do not use this option, the AVM does not skip a line between expressions unless the expressions do not fit on one line. If you use the SKIP option, but do not specify *n*, or if *n* is 0, the AVM starts a new line unless it is already at the beginning of a new line.

^

Tells the AVM to ignore an input field when input is being read from a file. Also, the following statement will read a line from an input file and ignore that line. This is an efficient way to skip over lines.

```
PROMPT-FOR ^
```

GO-ON (*key-label* . . .)

The GO-ON option tells the AVM to execute the GO action when the user presses any of the keys listed. The keys you list are used in addition to keys that perform the GO action by default (such as **F1** or **RETURN** on the last field) or because of ON statements.

When you list a key in the GO-ON option, you use the keyboard label of that key. For example, if you want the AVM to take the GO action when the user presses **F2**, you use the statement GO-ON(**F2**). If you list more than one key, separate them with spaces, not commas.

IN WINDOW *window*

Specifies the window in which the prompt occurs. The expression *window* must resolve to a handle to a window.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

editing-phrase

Supported only for backward compatibility.

Identifies processing to take place as each keystroke is entered. This is the syntax for *editing-phrase*:

```
[ label : ] EDITING: statement . . . END
```

For more information on *editing-phrase*, see the EDITING phrase reference entry.

record

The name of a record buffer. All of the fields in the record will be processed exactly as if you prompted for each of them individually.

To use PROMPT-FOR with a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

EXCEPT *field*

Affects all fields except those listed in the EXCEPT phrase.

Examples

The `r-prmpt.p` procedure requests a customer number from the user and stores that number in the screen buffer. The FIND statement reads a record from the Customer database table.

r-prmpt.p

```
REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum NO-ERROR.
  IF NOT AVAILABLE Customer THEN DO:
    MESSAGE "No such customer number.".
    UNDO, RETRY.
  END.
  DISPLAY Customer.Name Customer.Phone Customer.SalesRep.
END.
```

The `r-prmpt2.p` procedure requests the initials of a sales representative and stores those initials in the screen buffer. The FIND statement uses the initials stored in the screen buffer to read a record from the `SalesRep` database table. After finding the record, the procedure displays sales rep information.

r-prmpt2.p

```
REPEAT:
  PROMPT-FOR SalesRep.SalesRep LABEL "Sales rep's initials"
    WITH FRAME namefr ROW 2 SIDE-LABELS.
  FIND SalesRep NO-LOCK USING SalesRep.SalesRep.
  DISPLAY SalesRep.RepName SalesRep.Region SalesRep.MonthQuota
    WITH 1 DOWN NO-HIDE.
END.
```

Notes

- PROMPT-FOR puts user-supplied data into a screen buffer. It does not put any data into a record buffer. Therefore, if you want to use the data in the screen buffer, you must use the INPUT function to refer to the

data in the screen buffer or use the ASSIGN statement to move the data from the screen buffer into a record buffer. You can also use the USING option to FIND a record with the screen data index value.

- When ABL compiles a procedure, it designs all the frames used by that procedure. When it encounters a PROMPT-FOR statement, ABL designs the display of the prompt fields. When the procedure is run, the PROMPT-FOR statement puts data into those fields.
- If you are getting input from a device other than the terminal, and the number of characters read by the PROMPT-FOR statement for a particular field or variable exceeds the display format for that field or variable, the AVM returns an error. However, if you are setting a logical field that has a format of "y/n" and the data file contains a value of YES or NO, the AVM converts that value to "y" or "n".
- In the context of the .NET blocking method, `System.Windows.Forms.Application:Run( )`, if you execute the PROMPT-FOR statement for a non-modal ABL window from within a user-defined function or non-VOID method, this statement raises the STOP condition. For more information on the .NET `Application:Run( )` method, see the reference entry for the WAIT-FOR statement (.NET and ABL).

See also

DEFINE STREAM statement, EDITING phrase, Format phrase, Frame phrase, Stream object handle

PROMSGS function

Returns the current value of the ABL PROMSGS variable.

Syntax

```
PROMSGS
```

Example

This example uses the PROMSGS function to determine whether the default message file (promsgs) is in use. If not, it uses the PROMSGS function again to display the name of the current message file.

r-promsg.p

```
IF PROMSGS = "promsgs" THEN
  MESSAGE "Using default promsgs file.".
ELSE
  MESSAGE "Using" PROMSGS.
```

See also

PROMSGS statement

PROMSGS statement

Sets the ABL PROMSGS variable for the current ABL session. The PROMSGS variable holds the name of the current ABL message file. ABL supplies different versions of this file to support various languages.

Syntax

```
PROMSGS = string-expression
```

string-expression

A character-string expression that resolves to the name of an ABL message file. You can specify a full or relative pathname for the messages file. Any relative pathname is relative to your current working directory. In Windows, the pathname cannot contain characters outside of the non-Unicode code page. See *Internationalize ABL Applications* for more information about Unicode and code pages.

Example

This example prompts the user for a language name and then tries to find a message file for that language. If the message file is found, then the PROMSGS statement is used to make that the current message file. Subsequently, all ABL system messages are read from the new `promsgs` file. The PROMSGS function is used in an informative message.

r-swmsgs.p

```
DEFINE VARIABLE newlang AS CHARACTER NO-UNDO FORMAT "x(16)"
  LABEL "Language".
DEFINE VARIABLE msgfile AS CHARACTER NO-UNDO.

SET newlang HELP "Enter the new language for messages.".
IF newlang = "English" THEN msgfile = "promsgs".
ELSE msgfile = "prolang/promsgs." + LC(SUBSTRING(newlang, 1, 3)).

IF SEARCH(msgfile) <> ? THEN DO:
  PROMSGS = msgfile.
  MESSAGE "Messages will now be taken from" PROMSGS.
END.
ELSE DO:
  MESSAGE "Cannot find" msgfile.
  UNDO, RETRY.
END.
```

See also

PROMSGS function

PROPATH function

Returns the current value of the PROPATH environment variable.

Syntax

```
PROPATH
```

Example

This procedure first displays a comma-separated list of the directories in the current PROPATH. It then displays each directory in the current PROPATH, one per line.

r-ppath1.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

DISPLAY PROPATH.

REPEAT ix = 1 TO NUM-ENTRIES(PROPATH):
  DISPLAY ENTRY(ix , PROPATH) FORMAT "x(30)".
END.
```

Notes

- ABL stores the PROPATH as a comma-separated list of directories. (ABL strips the operating-specific separation characters (a colon (:) on UNIX; a semicolon (;) in Windows) and replaces them with commas.
- The default format for PROPATH is x(70).
- For more information on the PROPATH environment variable, see *Install OpenEdge* .

See also

PROPATH statement

PROPATH statement

Sets the PROPATH environment variable for the current ABL session.

When you start ABL, it automatically adds the $DLC directory and some subdirectories to your PROPATH. ABL always preserves these directories in your PROPATH, even if you change or clear your PROPATH. Thus, ABL can always find its executables and r-code.

Syntax

```
PROPATH = string-expression
```

string-expression

A field, variable, string constant, or combination of these that evaluates to a character string. The character string should be a list of directory paths. The directory names in the path can be separated by commas or by the appropriate separation character for your operating system. The directory pathnames can use the UNIX format for pathnames (`/dir1/dir2/dir3`, for example) or the standard pathname format for your operating system. Use the slash-separated directory name format if you are concerned about portability across multiple operating systems. In Windows, the pathnames cannot contain characters outside of the non-Unicode code page. See *Internationalize ABL Applications* for more information about Unicode and code pages.

Note: The += Addition assignment operator can be used with the PROPATH statement.

Examples

The r-ppath.p procedure displays a strip menu with four choices. The procedure defines three arrays: menu holds the items for selection on the menu, proglist holds the names of the programs associated with the menu selections, and ppath holds the appropriate PROPATHs for each program. The CHOOSE statement allows the user to choose an item from the strip menu.

r-ppath.p

```
DEFINE VARIABLE menu     AS CHARACTER NO-UNDO EXTENT 4 FORMAT "X(20)"
  INITIAL ["1. Sales","2. Acctg","3. Personnel","4. Exit"].
DEFINE VARIABLE proglist AS CHARACTER NO-UNDO EXTENT 4 FORMAT "X(8)"
  INITIAL ["sales.p","acctg.p","per.p","exit.p"].
DEFINE VARIABLE ppath    AS CHARACTER NO-UNDO EXTENT 4
  INITIAL ["sales/s-procs","acctg/a-procs","per/p-procs",","].

REPEAT:
  DISPLAY menu WITH TITLE " M A I N   M E N U " CENTERED
    1 COLUMN 1 DOWN NO-LABELS ROW 8.
  CHOOSE FIELD menu AUTO-RETURN.
  HIDE.
  PROPATH = ppath[FRAME-INDEX].
  RUN VALUE(proglist[FRAME-INDEX]).
END.
```

The AVM uses the menu selection number as an index into the ppath and proglist arrays. The AVM sets the PROPATH and runs the program.

This simple example changes and displays the PROPATH:

r-prpath.p

```
PROPATH = ENTRY(1,PROPATH) + ",/dlc,/dlc/proguide,/dlc/appl1/procs".
DISPLAY PROPATH.
```

The following example code uses the += assignment operator to add to the PROPATH.

```
PROPATH += ",/myNewAppDir1,/myNewAppDir2".
```

Notes

- Changes to PROPATH last only for the current session. Any subprocesses inherit the PROPATH in effect when the ABL session started.
- When you start ABL, it automatically adds the top directory of the ABL hierarchy and some subdirectories to your PROPATH. If you use the PROPATH statement to make a change, ABL adds the directories you specify to your existing PROPATH.
- ABL replaces separation characters in *expression* (a colon (:) on UNIX; a semicolon (;) in Windows) with commas, so the resulting PROPATH string can be accessed with the ENTRY function. Therefore, file pathnames passed in *expression* must not include embedded commas.
- If you change your PROPATH, and your old PROPATH included r-code libraries that are not in your new PROPATH, those libraries are automatically closed. If you run a procedure from a closed library, the AVM displays an error message.
- For more information on the PROPATH environment variable, see *Install OpenEdge*.

See also

ENTRY function, PROPATH function

PROVERSION function

Returns the version of ABL, or release of OpenEdge, you are running. You can also optionally retrieve additional release information, such as the service pack version number and the hot fix version number.

Syntax

```
PROVERSION [(mode)]
```

mode

An optional integer expression that identifies how much version information is returned, as follows:

- If the value of *mode* is either `0` or not specified, PROVERSION returns limited version information—the major version number, the minor version number, and, if applicable, the version tag.
- If the value of *mode* is `1`, PROVERSION returns full version information—the major version number, the minor version number, the service pack number, the hot fix version number, the build number, and, if applicable, the version tag.
- If the value of *mode* is other than `1` or `0`, a run-time error message is returned.

Examples

Following is an example of the information PROVERSION returns when you use `1` as the *mode* value for an FCS OpenEdge release, which has no version tag:

```
11.2.0.0.1234
```

in which:

- `11` is the major version number.
- `2` is the minor version number.
- `0` is the service pack number.
- `0` is the hot fix version number.
- `1234` is the build number.

If you specify `0` as the mode value (or specify no mode value at all) for an OpenEdge BETA release, you see the major version number, the minor version number, and the version tag. For example:

```
11.2BETA
```

If you specify 1 as the mode value for an OpenEdge BETA release, for example, the following details appear:

```
11.2.0.0.1234.BETA
```

See also

DBVERSION function, PROGRESS function

PUBLISH statement

Causes an ABL named event to occur.

Note: ABL named events are completely different from the key function, mouse, widget, and direct manipulation events described in the Handle-based Object Events Reference. They are also different from the class events described in the Class Events Reference.

Syntax

```
PUBLISH event-name
  [ FROM publisher-handle ]
  [ ( parameter [ , parameter ] ... ) ]
```

event-name

A quoted character string or character expression representing the name of a named event. If you use a quoted character string, ABL adds *event-name* to the PUBLISHED-EVENTS attribute's list of events.

FROM *publisher-handle*

A procedure or handle representing the procedure or widget to which ABL attributes the named event.

The FROM option lets a procedure publish an event on behalf of another procedure or widget. For example, if you want procedure A to publish a named event on behalf of procedure B, set *publisher-handle* to the procedure handle of B.

If the FROM option does not appear, ABL attributes the event to THIS-PROCEDURE, the procedure that contains the PUBLISH statement.

Note: If the FROM option does not appear and the PUBLISH statement occurs in a nonpersistent procedure that does not publicize its handle, potential subscribers have no way of knowing the handle's value, and can subscribe to the event only by using the SUBSCRIBE statement's ANYWHERE option.

(*parameter* [, *parameter*] ...)

The parameters, if any, of the named event.

As in the RUN statement, you must supply a value for each INPUT and INPUT-OUTPUT parameter and a variable for each OUTPUT parameter.

Also, if a named event has one or more parameters, the PUBLISH statement and each subscriber's local internal procedure (which the SUBSCRIBE statement names and which the AVM runs when the named event occurs) must specify identical signatures-where *signature* means the number of parameters and the data type and mode (INPUT, etc.) for each.

Note: When the named event occurs and the AVM runs each subscriber's local internal procedure, if the signature of a local internal procedure does not match the signature in the PUBLISH statement, the AVM reports a run-time error. Since the PUBLISH statement runs with an implicit NO-ERROR, errors are stored in the ERROR-STATUS handle.

The parameter syntax is identical to that of the RUN statement. For its specification, see the Parameter passing syntax reference entry.

Examples

The following example consists of four procedure files: a driver, a publisher, and two subscribers. The driver, `r-nedrvr.p`, runs the publisher and the two subscribers persistently, then subscribes to the event NewCustomer on behalf of the second subscriber.

r-nedrivr.p

```
/* r-nedrvr.p  */
DEFINE VARIABLE hPub  AS HANDLE NO-UNDO.
DEFINE VARIABLE hSub1 AS HANDLE NO-UNDO.
DEFINE VARIABLE hSub2 AS HANDLE NO-UNDO.

DEFINE BUTTON bNewCust LABEL "New Customer".
DEFINE BUTTON bQuit    LABEL "Quit".

RUN r-nepub.p  PERSISTENT SET hPub.
RUN r-nesub1.p PERSISTENT SET hSub1 (hPub).
RUN r-nesub2.p PERSISTENT SET hSub2.

/* Subscribe to event NewCustomer on behalf of subscriber 2 */
SUBSCRIBE PROCEDURE hSub2 TO "NewCustomer" IN hPub.

FORM bNewCust bQuit WITH FRAME x.

ENABLE ALL WITH FRAME x.

ON CHOOSE OF bNewCust RUN NewCust in hPub.

WAIT-FOR CHOOSE OF bQuit OR WINDOW-CLOSE OF CURRENT-WINDOW.
```

The publisher, `r-nepub.p`, publishes the event NewCustomer:

r-nepub.p

```
/* r-nepub.p */
PROCEDURE NewCust:
  DEFINE VARIABLE name AS CHARACTER NO-UNDO INITIAL "Sam".

  /* Let subscriber know new customer */
  PUBLISH "NewCustomer" (INPUT name).
END PROCEDURE.
```

The first subscriber, `nesub1.p`, subscribes to the event NewCustomer:

r-nesub1.p

```
/* r-nesub1.p */
DEFINE INPUT PARAMETER hPub AS HANDLE NO-UNDO.

SUBSCRIBE TO "NewCustomer" IN hPub.

PROCEDURE NewCustomer:
  DEFINE INPUT PARAMETER name AS CHARACTER NO-UNDO.

  MESSAGE "Subscriber 1 received event NewCustomer concerning" name
    VIEW-AS ALERT-BOX.
END.
```

The second subscriber, `nesub2.p`, already subscribed to the event NewCustomer, cancels all subscriptions:

r-nesub2.p

```
/* r-nesub2.p */
PROCEDURE NewCustomer:
  DEFINE INPUT PARAMETER name AS CHARACTER NO-UNDO.

  MESSAGE "Subscriber 2 received event NewCustomer concerning" name
    VIEW-AS ALERT-BOX.
  /* This subscriber receives the first event, then removes itself */
  UNSUBSCRIBE TO ALL.
END.
```

To start the example, run the driver, `r-nedrvr.p`.

Notes

- If a named event has multiple subscribers, the order in which the AVM notifies subscribers is undefined.
- INPUT-OUTPUT parameters can accumulate values from a set of subscribers. When a subscriber receives an INPUT-OUTPUT parameter, it has the value that the previous subscriber set it to. When the publisher receives an INPUT-OUTPUT parameter, it has the value that the last subscriber set it to.
- If a named event with multiple subscribers has OUTPUT parameters, each time a subscriber sets an OUTPUT parameter, the AVM overwrites the previous value. For this reason, Progress Software Corporation recommends that you use OUTPUT parameters with named events only when there is a single subscriber.
- If a named event has multiple subscribers and several subscribers specify a RETURN statement with a return value, the RETURN-VALUE function evaluates to the return value set by the last subscriber.
- The AVM executes the PUBLISH statement with an implicit NO-ERROR option. To find out if any errors occurred, and if so, which ones, use the ERROR-STATUS system handle.
- If *`publisher-handle`* is a handle, the value of SOURCE-PROCEDURE in each of the subscribers' internal procedures will be the handle of the procedure that created the widget.

See also

Publish( ) event method, PUBLISHED-EVENTS attribute, SUBSCRIBE statement, UNSUBSCRIBE statement

PUT CURSOR statement
(Character only)

Makes the cursor visible on the screen at a specified position.

In data-handling statements such as UPDATE, SET, PROMPT-FOR, and INSERT, the AVM handles cursor display so the user knows where the cursor is located in the window. However, if data is entered through the READKEY statement, and that statement is not part of an EDITING phrase, you might want to turn the cursor on so the user can see the location of the cursor while entering data.

Note: Does not apply to SpeedScript programming.

Syntax

```
PUT CURSOR
  {     OFF
    |{[ ROW expression ][ COLUMN expression ]}
  }
```

`OFF`

Ends display of the cursor.

`ROW` *`expression`*

The row in which you want to display the cursor. In the ROW option, *`expression`* is a constant, field name, variable name, or expression whose value is an integer that indicates the row where you want to display the cursor. If you do not use the ROW option, PUT CURSOR does not reposition the cursor. Similarly, if you specify a ROW that is outside the screen area, the AVM does not reposition the cursor.

`COLUMN` *`expression`*

The column in which you want to display the cursor. In the COLUMN option, *`expression`* is a constant, field name, variable name, or expression whose value is an integer that indicates the column where you want to display the cursor. If you do not use the COLUMN option, PUT CURSOR does not reposition the cursor. Similarly, if you specify a COLUMN that is outside the windows area, the AVM does not repositions the cursor.

Example

The following procedure uses PUT CURSOR to make the cursor visible in an editor window. When you run the procedure, you see a frame in a window. You can type text into this frame. The procedure reads each key you enter and takes the appropriate action. Then PUT CURSOR places the cursor in the first row and the first column in the editing frame when you first run the procedure. As you type, the cursor continues to be visible. As the procedure passes through the REPEAT loop for each keystroke, it takes action based on each keystroke and moves the cursor as it takes the action.

The procedure stores the information you type in the comments array, one character at a time. When you finish typing, press **GO**. The procedure displays the array where the AVM stored the typed information.

r-cursor.p

```
DEFINE VARIABLE comment AS CHARACTER NO-UNDO FORMAT "x(30)" EXTENT 4.
DEFINE VARIABLE iRow    AS INTEGER   NO-UNDO.
DEFINE VARIABLE iCol    AS INTEGER   NO-UNDO.
DEFINE VARIABLE lmargin AS INTEGER   NO-UNDO INITIAL 5.
DEFINE VARIABLE rmargin AS INTEGER   NO-UNDO INITIAL 34.
DEFINE VARIABLE ptop    AS INTEGER   NO-UNDO INITIAL 10.
DEFINE VARIABLE pbot    AS INTEGER   NO-UNDO INITIAL 13.
DEFINE VARIABLE r-ofst  AS INTEGER   NO-UNDO INITIAL 9.
DEFINE VARIABLE c-ofst  AS INTEGER   NO-UNDO INITIAL 4.

FORM SKIP(4) WITH WIDTH 32 ROW 9 COL 4 TITLE "Editor".

MESSAGE "Type text into the editor.  Press" KBLABEL("GO") "to end.".

VIEW.

ASSIGN
  iRow = ptop
  iCol = lmargin.

REPEAT:
  PUT CURSOR ROW iRow COLUMN iCol.
  READKEY.

  IF KEYFUNCTION(LASTKEY) = "GO" THEN LEAVE.
  IF KEYFUNCTION(LASTKEY) = "END-ERROR" THEN RETURN.
  IF LASTKEY = KEYCODE("CURSOR-RIGHT") THEN DO:
    iCol = iCol + 1.
    IF iCol > rmargin THEN iCol = lmargin.
    NEXT.
  END.
  IF LASTKEY = KEYCODE("CURSOR-LEFT") THEN DO:
    iCol = iCol - 1.
    IF iCol < lmargin THEN iCol = rmargin.
    NEXT.
  END.
  IF LASTKEY = KEYCODE("CURSOR-DOWN") THEN DO:
    iRow = iRow + 1.
    IF iRow > pbot THEN iRow = ptop.
    NEXT.
  END.
  IF LASTKEY = KEYCODE("CURSOR-UP") THEN DO:
    iRow = iRow - 1.
    IF iRow < ptop THEN iRow = pbot.
    NEXT.
  END.
  IF LASTKEY = KEYCODE("RETURN") THEN DO :
    iRow = iRow + 1.
    IF iRow > pbot THEN iRow = ptop.
    iCol = lmargin.
    NEXT.
  END.
  IF LASTKEY = KEYCODE("BACKSPACE") THEN DO:
    IF iCol = lmargin AND iRow = ptop THEN NEXT.
    iCol = iCol - 1.
    IF iCol < lmargin THEN DO:
      iCol = rmargin.
      iRow = iRow - 1.
      IF iRow < ptop THEN iRow = ptop.
    END.
    PUT SCREEN ROW iRow COLUMN iCol " ".
    OVERLAY(comment[iRow - r-ofst], iCol - c-ofst) = " ".
    NEXT.
  END.
  IF LASTKEY >= 32 AND LASTKEY <= 126 THEN DO:
    PUT SCREEN ROW iRow COLUMN iCol KEYLABEL(LASTKEY).
```

```
    OVERLAY(comment[iRow - r-ofst], iCol - c-ofst) = KEYLABEL(LASTKEY).
    iCol = iCol + 1.
    IF iCol > rmargin THEN DO:
      iCol = lmargin.
      iRow = iRow + 1.
      IF iRow > pbot THEN iRow = ptop.
    END.
  END.
END.

DISPLAY comment WITH FRAME x NO-LABELS
  TITLE "Comments Array" 1 COLUMN ROW 09 COLUMN 40.

MESSAGE "Information stored in the comments array.".
```

Notes

- You must use the PUT SCREEN statement to display data when you use the PUT CURSOR statement. You also have to define a variable for the cursor position, and increment it as the AVM reads the keys entered by the user if you want the cursor to move as the user types.
- The PUT CURSOR statement displays the cursor until you use the PUT CURSOR OFF statement to stop the display.
- Because a cursor is always displayed in an EDITING phrase, using the PUT CURSOR statement in an EDITING phrase (or if you have not issued a PUT CURSOR OFF statement before the phrase) might cause errors.

See also

PUT SCREEN statement

PUT SCREEN statement
(Character only)

Displays a character expression at a specified location on a screen, overlaying any other data that might be displayed at that location.

This statement is supported only for backward compatibility.

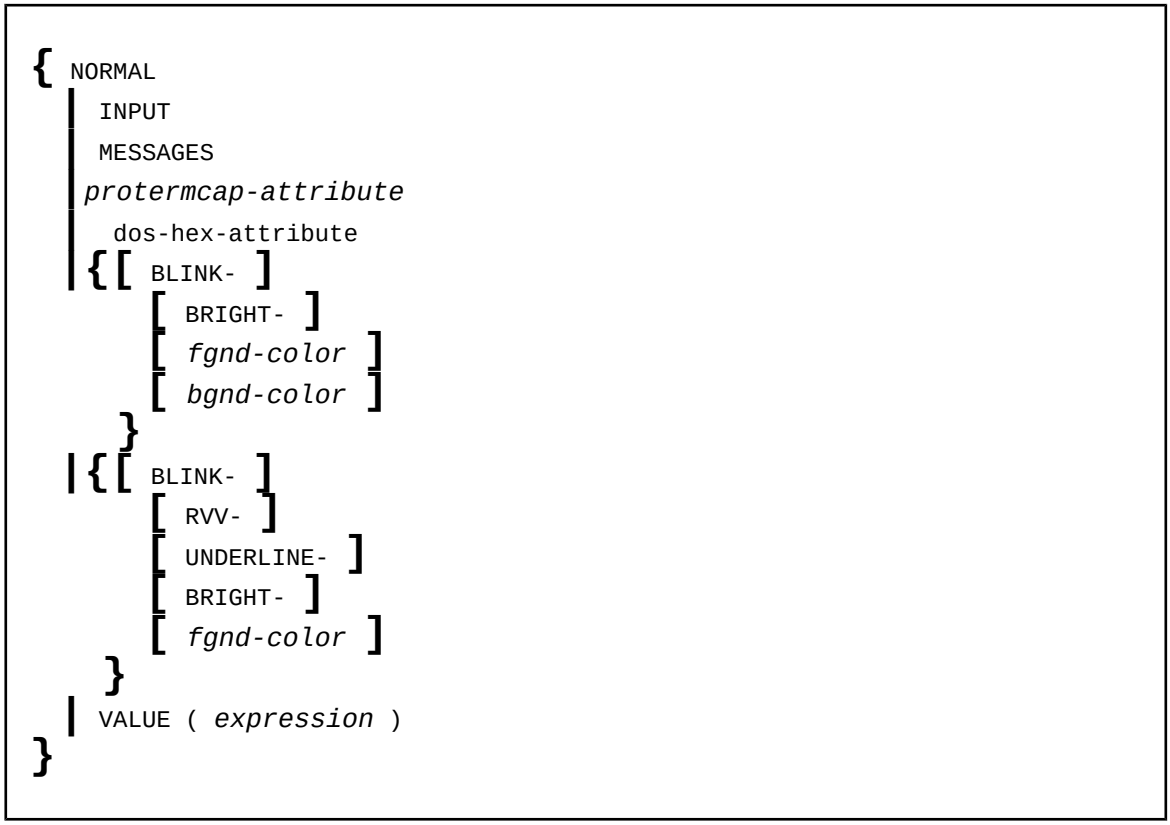
Note: Does not apply to SpeedScript programming.

Syntax

```
PUT SCREEN
   [ COLOR color-phrase ]
   [ COLUMN expression ]
   [ ROW expression ]
  expression
```

COLOR *color-phrase*

The video attributes you want to use to display an expression. When you display data in the first column of a spacetaking terminal, the AVM does not display that data with color. If you are displaying data in a column other than column 1, the AVM displays the color attribute in the column prior to the current column (current column minus 1).



For more information, see the COLOR phrase reference entry.

COLUMN *expression*

The column in which you want to display an expression. In the COLUMN option, *expression* is a constant, field name, variable name, or expression whose value is an integer that indicates the column in which you want to display an expression. If you do not use the COLUMN option, PUT SCREEN displays the expression at column 1. If you specify a COLUMN that is outside the screen area, the AVM disregards the PUT SCREEN statement.

ROW *expression*

The row in which you want to display an expression. In the ROW option, *expression* is a constant, field name, variable name, or expression whose value is an integer that indicates the row you want to display an expression. If you do not use the ROW option, PUT SCREEN displays the expression at row 1. If you specify a ROW that is outside the screen area, the AVM disregards the PUT SCREEN statement.

expression

A constant, field name, variable name, or expression that results in a character string. The character string can contain control characters and can be as long as you want.

Example

The `r-putscr.p` procedure determines whether a Customer's current balance is above or below 0. If it is above 0, they have a credit; if it is below 0, they owe money. The label of the balance column is changed based on whether they have a credit or owe money.

r-putscr.p

```
DEFINE VARIABLE paid-owed AS DECIMAL   NO-UNDO.
DEFINE VARIABLE bal-label AS CHARACTER NO-UNDO FORMAT "x(20)".

FOR EACH Customer NO-LOCK:
  paid-owed = Customer.Balance.
  IF paid-owed < 0 /* Customer has a credit */ THEN DO:
    paid-owed = - paid-owed.
    bal-label = "Customer Credit".
  END.
  ELSE bal-label = "Unpaid balance".

  DISPLAY Customer.CustNum Customer.Name paid-owed LABEL " " WITH 1 DOWN.
  IF Customer.Balance < 0 THEN
    PUT SCREEN COLOR MESSAGES ROW 2 COLUMN 34 bal-label.
  ELSE
    PUT SCREEN ROW 2 COLUMN 34 bal-label.
END.
```

If the Customer has a credit (balance < 0) the first PUT SCREEN statement displays the value of bal-label (which is Customer Credit) in the same color as you see system MESSAGES (usually reverse video).

If the Customer owes money (balance > 0) the second PUT SCREEN statement displays the value of bal-label (which is Current Balance) in normal display mode.

Notes

- Values displayed by PUT SCREEN are not the same as values that belong to frames. Thus those expressions can be overwritten by other displays or hides. Ensure that values displayed by PUT SCREEN do not overwrite frame fields that are used later for data entry.
- If you use the PUT SCREEN statement in a procedure that runs in batch or background mode, the AVM disregards the PUT SCREEN statement.
- The HIDE ALL statement clears the entire screen, including any data displayed by a PUT SCREEN statement.
- If you use the PUT SCREEN statement to display data in the message area, the HIDE MESSAGES statement does not necessarily clear that data.

See also

COLOR phrase, DISPLAY statement, HIDE statement, PUT statement

PUT statement

Sends the value of one or more expressions to an output destination other than the terminal.

Syntax

```
PUT
   [ STREAM stream | STREAM-HANDLE handle ]
   [ UNFORMATTED ]
   [{expression
        [ FORMAT string ]
        [{ AT | TO }expression ]
     }
     | SKIP [ ( expression ) ]
     | SPACE [ ( expression ) ]
   ]...
```

```
PUT [ STREAM stream | STREAM-HANDLE handle ] CONTROL expression...
```

`STREAM` *name*

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream. See the DEFINE STREAM statement reference entry and topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams.

`STREAM-HANDLE` *handle*

Specifies the handle to a stream. If *handle* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

`UNFORMATTED`

Tells the AVM to display each expression in the same format produced by the EXPORT statement, but without quotes.

expression

Specifies a constant, field name, variable name, or expression.

`FORMAT` *string*

The format in which you want to display the *expression*. If you do not use the FORMAT option, the AVM uses the defaults shown in the following table.

Table 57: Default display formats

Type of expression	Default format
Field	Format from Dictionary
Variable	Format from variable definition

Type of expression	Default format
Constant character	Length of character string
Other	Default format for the data type of the expression

The following table shows the default formats for other expressions.

Table 58: Default data type display formats

Data type	Default display format
CHARACTER	x(8)
CLASS and ENUM	>>>>>>9
DATE	99/99/99
DATETIME	99/99/9999 HH:MM:SS.SSS
DATETIME-TZ	99/99/9999 HH:MM:SS.SSS+HH:MM
DECIMAL	->>,>>9.99
HANDLE	>>>>>>9
INT64	->,>>>,>>9
INTEGER	->,>>>,>>9
LOGICAL	yes/no
LONGCHAR[57]	See footnote.
MEMPTR[58]	See footnote.
RAW[58]	See footnote.
RECID	>>>>>>9
ROWID[58]	See footnote.

AT *expression*

Specifies the column position where you want to place the output value. If that position has already been used on the current line, PUT skips to the next line and puts the *expression* in the specified column.

[57] You can only display a LONGCHAR using the VIEW-AS EDITOR LARGE phrase.

[58] You cannot display a MEMPTR, RAW, or ROWID value directly. However, you can convert it to a character string representation using the STRING function and display the result. A ROWID value converts to a hexadecimal string, "0x*hexdigits*", where *hexdigits* is any number of characters "0" through "9" and "A" through "F". A MEMPTR or RAW value converts to decimal integer string.

TO *expression*

Specifies the column position where you want to end the output value being output. If that position has already been used on the current line, PUT skips to the next line and puts the *expression* in the specified column.

SKIP [(*expression*)]

Specifies the number of new lines you want to output. If you do not use the SKIP option, PUT will not start a new line to the output stream. If you use the SKIP parameter, but do not specify *expression* (or if *expression* is 0), the AVM starts a new line only if output is not already positioned at the beginning of a new line.

SPACE [(*expression*)]

Specifies the number of spaces you want to output. Spaces are not placed between items being PUT unless you use the SPACE option.

CONTROL *expression*

The expression specifies a control sequence that you want to send without affecting the current line, page counters, and positions maintained within ABL. Following CONTROL, *expression* can be a character-string expression or a RAW variable. It can include null character constants of the form NULL or NULL(*expression*), where *expression* specifies the number of NULLs to send. See the Notes section in this reference entry for details.

Example

This procedure creates a text file that contains the names of each customer. The names are separated from each other by a slash (/). The entire file consists of one long line.

r-put.p

```
DEFINE STREAM s1.

OUTPUT STREAM s1 TO cus.dat.

FOR EACH Customer NO-LOCK:
  PUT STREAM s1 name "/".
END.

OUTPUT STREAM s1 CLOSE.
```

Notes

- In the AT, TO, SKIP, and SPACE options, if *expression* is less than or equal to 0, the AVM disregards the option.
- The PUT statement never automatically starts a new line. You must use SKIP to explicitly start a new line.

- The PUT statement uses the default display format for the data type of the field or variable you name in the PUT statement. The PUT statement does not overwrite an area that is already used by a previous format when it displays data. For example:

```
DEFINE VARIABLE myname AS CHARACTER NO-UNDO FORMAT "x(8)".
DEFINE VARIABLE mynum  AS CHARACTER NO-UNDO FORMAT "x(8)".

ASSIGN
  myname = "abc"
  mynum  = "123".

OUTPUT TO myfile.
PUT myname AT 8 mynum AT 12.
OUTPUT CLOSE.
```

```
abc
123
```

  Use the UNFORMATTED option with the PUT statement to override the format-sensitive display.

- You can use the NULL keyword to output null characters (\0) in a control sequence. For example, the following statements write the control sequence **ESC A** \0 and 20 NULLs to output stream **A**:

```
PUT STREAM A CONTROL "~033A" NULL.
PUT STREAM A CONTROL NULL(20).
```

- You can use the PUT statement with an object reference for a class or enum instance. The PUT statement implicitly calls the ToString( ) method of the class to convert the specified object reference to a character value before it sends the value to the output destination. In the case of an enum instance, the character value will be the appropriate enumeration member or a comma-delimited list of members in the case of a flag enum with multiple flags set.

See also

DEFINE STREAM statement, DISPLAY statement, EXPORT statement, OUTPUT TO statement, PAGE statement, PUT SCREEN statement, Stream object handle

PUT-BITS statement

Uses the bit representation of an integer to set a given number of bits at a given location within another integer.

Syntax

```
PUT-BITS ( destination , position , numbits ) = expression
```

destination

An ABL integer variable. The statement sets bits in *destination* that correspond to the bits that are on in the source variable, *expression*. It clears bits in the destination variable that are 0 in the source variable. Note that the number of bits set or cleared is limited by the *numbits* parameter, and the location within the destination is determined by the *position* variable.

position

A variable or expression that returns an integer. This parameter designates the position of the lowest-order bit of the bits that are to be interpreted as an integer. Bits are numbered from 1 through the length of an integer; with 1 being the low-order bit. If *position* is greater than the length of an integer or less than 1, the AVM generates a run-time error.

numbits

The number of bits to examine when generating the return value. If *position* plus *numbits* is greater than the length of an integer plus 1, the AVM generates a run-time error.

expression

A source variable that returns an integer. If the integer cannot be represented in the number of bits specified by numbits, the AVM stores the low-order numbits bits of the integer.

See also

GET-BITS function

PUT-BYTE statement

Stores the unsigned 1-byte value of an integer expression at the specified memory location.

Syntax

```
PUT-BYTE ( destination , position ) = expression
```

destination

A variable of type RAW or MEMPTR. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and has not had its region allocated (by a SET-SIZE statement or by a Windows dynamic link library (DLL) or UNIX shared library routine), the AVM generates a run-time error.

position

An integer value greater than 0 that indicates the byte position where the AVM stores *expression*. If *position* is less than 1, the AVM generates a run-time error. For a RAW *destination*, if *position* is greater than the length of *destination*, the AVM changes the length of *destination* to *position* and pads the gap with null bytes. For a MEMPTR *destination*, if *position* is greater than the length of *destination*, the AVM generates a run-time error.

expression

The integer value of a constant, field, variable, function, or expression. If *expression* is less than 0 or greater than 255, the AVM stores the right-most byte value of *expression* in *destination*.

Examples

This procedure finds the name of Customer 26, Jack's Jacks, and stores it in the RAW variable r1. The PUT-BYTE statement replaces the first four bytes in the Name with the specified character code values. The procedure then writes the values in r1 back into the Name field and displays that field. Jack's Jacks becomes Bill's Jacks.

r-rawput.p

```
/* You must connect to a non-OpenEdge demo database to run this procedure */
DEFINE VARIABLE r1 AS RAW NO-UNDO.

FIND Customer WHERE Customer.CustNum = 26.
DISPLAY Customer.Name.

ASSIGN
  r1                 = RAW(Customer.Name)
  PUT-BYTE(r1,1)     = ASC('B')
  PUT-BYTE(r1,2)     = ASC('i')
  PUT-BYTE(r1,3)     = ASC('l')
  PUT-BYTE(r1,4)     = ASC('l')
  RAW(Customer.Name) = r1.

DISPLAY Customer.Name.
```

The following example allocates a MEMPTR region large enough to hold the character string "Bill", terminated by a null byte. It stores the string one byte at a time using the PUT-BYTE statement, and then displays the string directly from the region.

r-mptput.p

```
DEFINE VARIABLE mptr AS MEMPTR NO-UNDO.

ASSIGN
  SET-SIZE(mptr)   = LENGTH("Bill") + 1
  PUT-BYTE(mptr,1) = ASC('B')
  PUT-BYTE(mptr,2) = ASC('i')
  PUT-BYTE(mptr,3) = ASC('l')
  PUT-BYTE(mptr,4) = ASC('l')
  PUT-BYTE(mptr,5) = 0.

DISPLAY GET-STRING(mptr,1).
```

Note

For more information on accessing DLL routines from ABL, see *OpenEdge Programming Interfaces*.

See also

GET-BYTE function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

PUT-BYTES statement

Copies a RAW or MEMPTR variable to the specified location in another RAW or MEMPTR variable.

Syntax

```
PUT-BYTES ( destination , position ) = expression
```

destination

An expression that returns a target RAW or MEMPTR variable. If *destination* is the Unknown value (?), PUT-BYTES does nothing.

position

An integer value greater than 0 that indicates the byte position where you want to put the data. If *position* is less than 1, the AVM generates a run-time error.

For a RAW variable, if *position* is greater than the length of *destination*, the AVM increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR variable, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, the AVM generates a run-time error. If *destination* is a RAW and *position* plus the length of *expression* is greater than 32K, the AVM generates a run-time error.

expression

An expression that returns a RAW or MEMPTR variable.

See also

GET-BYTES function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

PUT-DOUBLE statement

Stores the 8-byte floating-point value of a DECIMAL expression at the specified memory location.

Syntax

```
PUT-DOUBLE ( destination , position ) = expression
```

destination

A variable of type RAW or MEMPTR. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and has not had its region allocated (by a

SET-SIZE statement or by a Windows dynamic link library (DLL) or UNIX shared library routine), the AVM generates a run-time error.

position

An integer value greater than 0 that indicates the byte position where the AVM stores *expression*. If *position* is less than 1, the AVM generates a run-time error.

For a RAW *destination*, if *position* is greater than the length of *destination*, the AVM increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR *destination*, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, the AVM generates a run-time error.

expression

The DECIMAL value of a constant, field, variable, function, or expression.

Example

For examples of how to use the PUT-DOUBLE statement, see the PUT-BYTE statement reference entry.

Notes

- This statement supports byte-swapping only if *destination* is a MEMPTR data type. The statement will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately while putting the data into the MEMPTR memory.
- For more information on accessing DLL routines from ABL, see *OpenEdge Programming Interfaces*.

See also

GET-DOUBLE function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

PUT-FLOAT statement

Stores the 4-byte floating-point value of a DECIMAL expression at the specified memory location.

Syntax

```
PUT-FLOAT ( destination , position ) = expression
```

destination

A variable of type RAW or MEMPTR. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and has not had its region allocated (by a SET-SIZE statement or by a Windows dynamic link library (DLL) or UNIX shared library routine), the AVM generates a run-time error.

position

An integer value greater than 0 that indicates the byte position where the AVM stores *expression*. If *position* is less than 1, the AVM generates a run-time error.

For a RAW *destination*, if *position* is greater than the length of *destination*, the AVM increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR *destination*, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, the AVM generates a run-time error.

expression

The DECIMAL value of a constant, field, variable, function, or expression.

Example

For examples of how to use the PUT-FLOAT statement, see the PUT-BYTE statement reference entry.

Notes

- This statement supports byte-swapping only if *destination* is a MEMPTR data type. The statement will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately while putting the data into the MEMPTR memory.
- For more information on accessing DLL routines from ABL, see *OpenEdge Programming Interfaces*.

See also

GET-FLOAT function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

PUT-INT64 statement

Stores the signed 64-bit value of an INT64 or INTEGER expression at the specified memory location as an INT64 data type.

Syntax

```
PUT-INT64 ( destination , position ) = expression
```

destination

A variable of type RAW or MEMPTR. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and its region in not allocated (by a SET-SIZE statement or by a Windows dynamic link library (DLL) or UNIX shared library routine), the AVM generates a run-time error.

position

An integer value greater than 0 that indicates the byte position where the AVM stores *expression*. If *position* is less than 1, the AVM generates a run-time error.

For a RAW *destination*, if *position* is greater than the length of *destination*, the AVM increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR *destination*, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, the AVM generates a run-time error.

expression

The integer value of a constant, field, variable, function, or expression.

Example

See the example in the GET-INT64 function section.

See also

GET-INT64 function, INT64 function, PUT-LONG statement

PUT-KEY-VALUE statement
(Windows only)

Adds, modifies, and deletes keys in the current environment.

Note: Does not apply to SpeedScript programming.

Syntax

```
PUT-KEY-VALUE
  {{ SECTION section-name
       KEY {key-name | DEFAULT }
       VALUE value
     }
     |{ COLOR | FONT }{ number | ALL }
  }
  [ NO-ERROR ]
```

SECTION *section-name*

A CHARACTER expression that specifies the name of the section that contains the key of interest.

In initialization files, section names appear in square brackets([]). When you specify a section name in a PUT-KEY-VALUE statement, omit the square brackets.

KEY *key-name*

A CHARACTER expression that specifies the name of the key of interest.

DEFAULT

Tells PUT-KEY-VALUE to use the default key of section *section-name*.

Some applications store data in the registry under the default key of a section. This option lets you modify this data. For an example, see the EXAMPLES section of this entry.

This option applies only to the registry and not to initialization files.

VALUE *value*

The value of the key to write to the environment. *value* must evaluate to a CHARACTER expression of no more than 128 bytes.

COLOR { *number* | ALL }

Updates color definitions in the current environment from the definitions in the internal color table. The *number* parameter is a literal integer that specifies the number of a single color in the current environment whose definition you want to update. The ALL option updates all color definitions in the current environment.

FONT { *number* | ALL }

Updates font definitions in the current environment from the definitions in the internal font table. The *number* parameter is a literal integer that specifies the number of a single font in the current environment whose definition you want to update. The ALL option updates all font definitions in the current environment.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Examples

If the current environment resides in the registry, the PUT-KEY-VALUE statement:

1. Searches in the registry under the current environment for the subkey MYSECTION
2. Creates MYSECTION if it does not exist
3. Searches MYSECTION for the subkey MYKEY
4. Sets MYKEY to the value MYVARIABLE (if MYKEY exists), or adds MYKEY and the value MYVARIBLE (if MYKEY does not exist)

If the current environment resides in an initialization file, the PUT-KEY-VALUE statement:

1. Searches the initialization file for the section MYSECTION
2. Creates MYSECTION if it does not exist
3. Searches MYSECTION for the key MYKEY
4. Sets MYKEY to the value MYVARIABLE (if MYKEY exists), or adds MYKEY and the value MYVARIBLE (if MYKEY does not exist):

```
PUT-KEY-VALUE SECTION "MYSECTION" KEY "MYKEY" VALUE MYVARIABLE
```

If the current environment resides in the registry, the following examples add, directly under the current environment, the value name MYKEY and the value MYVARIABLE:

```
PUT-KEY-VALUE SECTION "" KEY "MYKEY" VALUE MYVARIABLE
```

```
PUT-KEY-VALUE SECTION "?" KEY "MYKEY" VALUE MYVARIABLE
```

If the current environment resides in an initialization file, the previous examples return an error.

If the current environment resides in the registry, the following examples:

1. Search in the registry under the current environment for the key MYSECTION.
2. Search MYSECTION for the value name MYKEY.
3. Delete MYKEY and its value.

```
PUT-KEY-VALUE SECTION "MYSECTION" KEY "MYKEY" VALUE ""
```

```
PUT-KEY-VALUE SECTION "MYSECTION" KEY "MYKEY" VALUE ?
```

If the current environment resides in an initialization file, the previous examples delete the key MYKEY, including its value, from the section MYSECTION.

If the current environment resides in the registry, the following examples delete the subkey MYSECTION, all values under MYSECTION, all subkeys under MYSECTION, and all values under those subkeys:

```
PUT-KEY-VALUE SECTION "MYSECTION " KEY "?" VALUE ?
```

```
PUT-KEY-VALUE SECTION "MYSECTION " KEY "" VALUE ""
```

If the current environment resides in an initialization file, the previous examples remove the section MYSECTION, and all key-value pairs within MYSECTION, from the initialization file.

If the current environment resides in the registry, the following example:

1. Searches the current environment for the subkey MYAPP
2. Sets the default key under MYAPP to NEWVALUE

```
PUT-KEY-VALUE SECTION "MYAPP" KEY DEFAULT VALUE "NEWVALUE"
```

If the current environment resides in an initialization file, the previous example returns an error.

Notes

- Environments typically consist of sections, each of which contains keys, each of which consists of a name and a value. A typical section name is COLORS. A typical key within this section consists of the name COLOR7 and the value 255,255,0. This key attaches the name COLOR7 to color value 255,255,0 (a color specification that uses the red-green-blue color-naming scheme).

  The current environment might be the registry or an initialization file. The registry consists of sections called keys and subkeys arranged in a hierarchy. Keys and subkeys contain value entries, each of which consists of a value name and value data. Initialization files, by contrast, consist of a single level of sections. Sections contain entries, each of which consists of a name, an equal sign (=), and a value.

  For more information on environments, see the topics on colors and fonts in *OpenEdge Programming Interfaces*.

- The current environment is one of the following:
  - The default environment
  - An environment that a startup parameter specified (the *startup environment*)
  - An environment that a LOAD statement loaded and that the most recent USE statement made current
- If you UNLOAD the current environment, a subsequent PUT-KEY-VALUE writes to the startup environment.
- To remove a key-value pair from an environment, set *key-name* to the name of the key and *value* to the Unknown value (?).
- To remove a section, including all its key-value pairs, from an environment, set *section-name* to the name of the section and *key-name* to the Unknown value (?).
- To change the definitions in the internal color table, use one of the following techniques:
  - To display a dialog box that lets the user change the color definitions, use the SYSTEM-DIALOG-COLOR statement.
  - To change the color definitions directly from ABL, use the attributes and methods of the COLOR-TABLE handle.

  Note: The COLOR option of the PUT-KEY-VALUE statement does not change the definitions in the internal color table. This option merely moves some or all of those definitions to the current environment.

- To change the definitions in the internal font table, use one of the following techniques:
  - To display a dialog box that lets the user change the font definitions, use the SYSTEM-DIALOG-FONT statement.
  - To change the font definitions directly from ABL, use the attributes and methods of the FONT-TABLE handle.

  Note: The FONT option of the PUT-KEY-VALUE statement does not change the definitions in the internal font table. This option merely moves some or all of those definitions to the current environment.

- For more information on colors and fonts, see the topics on colors and fonts in *OpenEdge Programming Interfaces*.

See also

COLOR-TABLE system handle, FONT-TABLE system handle, GET-KEY-VALUE statement, LOAD statement, NO-ERROR option, SYSTEM-DIALOG COLOR statement, SYSTEM-DIALOG FONT statement, UNLOAD statement, USE statement

PUT-LONG statement

Stores the signed 32-bit value of an integer expression at the specified memory location.

Syntax

```
PUT-LONG ( destination , position ) = expression
```

destination

A variable of type RAW or MEMPTR. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and its region in not allocated (by a SET-SIZE statement or by a Windows dynamic link library (DLL) or UNIX shared library routine), the AVM generates a run-time error.

position

An integer value greater than 0 that indicates the byte position where the AVM stores *expression*. If *position* is less than 1, the AVM generates a run-time error.

For a RAW *destination*, if *position* is greater than the length of *destination*, the AVM increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR *destination*, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, the AVM generates a run-time error.

expression

The integer value of a constant, field, variable, function, or expression.

Notes

- For examples of how to use the PUT-LONG statement, see the PUT-BYTE statement reference entry.
- This statement supports byte-swapping only if *destination* is a MEMPTR data type. The statement will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately while putting the data into the MEMPTR memory.
- For more information on accessing DLL routines from ABL, see *OpenEdge Programming Interfaces*.

See also

GET-LONG function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

PUT-SHORT statement

Stores the signed 16-bit value of an integer expression at the specified memory location.

Syntax

```
PUT-SHORT ( destination , position ) = expression
```

destination

A variable of type RAW or MEMPTR. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and its region is not allocated (by a SET-SIZE statement or by a Windows dynamic link library (DLL) or UNIX shared library routine), the AVM generates a run-time error.

position

An integer value greater than 0 that indicates the byte position where the AVM stores *expression*. If *position* is less than 1, the AVM generates a run-time error.

For a RAW *destination*, if *position* is greater than the length of *destination*, the AVM increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR *destination*, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, the AVM generates a run-time error.

expression

The integer value of a constant, field, variable, function, or expression.

Notes

- For examples of how to use the PUT-SHORT statement, see the PUT-BYTE statement reference entry.
- This statement supports byte-swapping only if *destination* is a MEMPTR data type. The statement will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately while putting the data into the MEMPTR memory.
- For more information on accessing DLL routines from ABL, see *OpenEdge Programming Interfaces*.

See also

GET-SHORT function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

PUT-STRING statement

Stores the null-terminated value of a CHARACTER or LONGCHAR expression at the specified memory location. If *numbytes* is specified, PUT-STRING will copy the requested number of bytes from the variable, regardless of whether there are embedded nulls. In this case PUT-STRING will not put a terminating null into the MEMPTR unless the last byte copied happens to be a null.

Syntax

```
PUT-STRING ( destination , position , [numbytes] ) = expression
```

destination

A variable of type RAW or MEMPTR. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and its region is not allocated (by a SET-SIZE statement or by a Windows dynamic link library (DLL) or UNIX shared library routine), the AVM generates a run-time error.

position

An integer value greater than 0 that indicates the byte position where the AVM stores *expression*. If *position* is less than 1, the AVM generates a run-time error.

For a RAW *destination*, if *position* is greater than the length of *destination*, the AVM increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR *destination*, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, the AVM generates a run-time error.

numbytes

An integer value greater than 0 that indicates how many bytes to copy from *expression.* If *position* plus *numbytes* is greater than the length of *destination*, the AVM generates a run-time error.

expression

An expression (a constant, field name, variable name, or expression) whose value is a CHARACTER or LONGCHAR. The AVM converts a LONGCHAR value to `-cpinternal` before it stores the value.

Notes

- For examples of how to use the PUT-STRING statement, see the PUT-BYTE statement reference entry.
- For more information on accessing DLL and UNIX shared library routines from ABL, see *OpenEdge Programming Interfaces*.

See also

GET-STRING function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

PUT-UNSIGNED-LONG statement

Writes an INTEGER or INT64 value as a 32-bit unsigned value to a MEMPTR or RAW value. This statement is analogous to the PUT-UNSIGNED-SHORT statement, except with a 32-bit value.

Syntax

```
PUT-UNSIGNED-LONG ( destination , position ) = expression
```

destination

A MEMPTR or RAW variable. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and its region is not allocated (by a SET-SIZE statement, or by a Windows DLL or Unix shared object entry point), the AVM generates a run-time error.

position

An INTEGER or INT64 value greater than 0 that indicates the byte position to write *expression*. If *position* is less than 1, the AVM generates a run-time error. For a RAW *destination*, if *position* is greater than the length of *destination*, the AVM increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes. For a MEMPTR *destination*, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, the AVM generates a run-time error.

For a RAW *destination*, if *position* is greater than the length of *destination*, the AVM increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR *destination*, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, the AVM generates a run-time error.

expression

An INTEGER or INT64 value of a constant, field, variable or expression. If *expression* is an INT64 value exceeding the maximum value of an UNSIGNED-LONG, the AVM generate an overflow error at run time.

Notes

- This statement supports byte-swapping only if *destination* is a MEMPTR data type. The statement will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately while putting the data into the MEMPTR memory.
- For more information on accessing DLL routines from ABL, see *OpenEdge Programming Interfaces*.

See also

GET-UNSIGNED-LONG function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

PUT-UNSIGNED-SHORT statement

Stores the unsigned 16-bit value of an integer expression at the specified memory location.

Syntax

```
PUT-UNSIGNED-SHORT ( destination , position ) = expression
```

destination

A variable of type RAW or MEMPTR. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and its region is not allocated (by a SET-SIZE statement or by a Windows dynamic link library (DLL) or UNIX shared library routine), the AVM generates a run-time error.

position

An integer value greater than 0 that indicates the byte position where the AVM stores *expression*. If *position* is less than 1, the AVM generates a run-time error.

For a RAW *destination*, if *position* is greater than the length of *destination*, the AVM increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR *destination*, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, the AVM generates a run-time error.

expression

The integer value of a constant, field, variable, function, or expression.

Notes

- This statement supports byte-swapping only if *destination* is a MEMPTR data type. The statement will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately while putting the data into the MEMPTR memory.
- For more information on accessing DLL routines from ABL, see *OpenEdge Programming Interfaces*.

See also

GET-UNSIGNED-SHORT function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

QUERY-OFF-END function

Returns a LOGICAL value indicating whether the specified query is positioned at the end of its result list (either before the first record or after the last record).

Syntax

```
QUERY-OFF-END ( query-name )
```

query-name

A character expression that evaluates to the name of a currently open query. If *query-name* does not resolve to the name of a query, or if the query is not open, then the function returns the Unknown value (?).

Note: Searching for a query using a handle is more efficient than a character expression. The AVM resolves a character expression at run time by searching in the current routine for a static query with that name. If not found, the AVM searches the enclosing main procedure. If still not found, the AVM searches up through the calling programs of the current routine, and their main procedures. Since a handle uniquely identifies a query, no such search is required. Use the query object handle's QUERY-OFF-END attribute to avoid a run-time search.

Example

The following example uses the QUERY-OFF-END function to determine when to leave the REPEAT loop:

r-qoff.p

```
OPEN QUERY cust-query FOR EACH Customer.

REPEAT:
  GET NEXT cust-query.
  IF QUERY-OFF-END("cust-query") THEN LEAVE.
  DISPLAY Customer.CustNum Customer.Name.
END.
```

When you run this procedure, all Customer numbers and names are displayed. After the last record is displayed, the loop iterates and the GET NEXT statement reads beyond the last record. At this point QUERY-OFF-END returns TRUE and the AVM exits the loop.

Note

To test whether a GET statement read beyond the last (or first) record, you can use the AVAILABLE function with the buffer name. You can also use the QUERY-OFF-END function, which serves the same purpose, but does not require a specific buffer; it requires only a query name.

See also

CLOSE QUERY statement, CURRENT-RESULT-ROW function, DEFINE BROWSE statement, DEFINE QUERY statement, GET statement, NUM-RESULTS function, OPEN QUERY statement, QUERY-OFF-END attribute, REPOSITION statement

QUERY-TUNING phrase

Allows programmatic control over the execution of a query in a DataServer application. This phrase is available for the DataServers; it is not available for queries of OpenEdge databases.

Syntax

```
QUERY-TUNING
  (
    {[ ARRAY-MESSAGE | NO-ARRAY-MESSAGE ]
       [ BIND-WHERE | NO-BIND-WHERE ]
       [ CACHE-SIZE integer ]
       [ DEBUG { SQL | EXTENDED diag-option }| NO-DEBUG ]
       [ INDEX-HINT | NO-INDEX-HINT ]
       [ JOIN-BY-SQLDB | NO-JOIN-BY-SQLDB ]
       [ LOOKAHEAD | NO-LOOKAHEAD ]
       [ ORDERED-JOIN ]
       [ REVERSE-FROM ]
       [ SEPARATE-CONNECTION | NO-SEPARATE-CONNECTION ]
    }
  )
```

The following descriptions are general. For more detailed information, see the OpenEdge DataServer Guides (*Use the Microsoft SQL Data Server* and *Use the Oracle Data Server*).

`ARRAY-MESSAGE | NO-ARRAY-MESSAGE`

Specifies whether the DataServer sends multiple result rows in a single logical network message.

The default is ARRAY-MESSAGE.

`BIND-WHERE | NO-BIND-WHERE`

This option is available only for the DataServer for ORACLE.

Specifies whether the DataServer uses ORACLE bind variables or literals in WHERE clauses. If you use NO-BIND-WHERE, the DataServer uses literals. Bind variables can improve performance, but ORACLE produces some unexpected results for some data types.

The default is BIND-WHERE.

`CACHE-SIZE integer [ ROW | BYTE ]`

Specifies the maximum cache size the DataServer can use when fetching records for a lookahead or standard cursor. You can optionally specify the size of the cache information in either bytes or records. The following values are for ORACLE.

The default is 1024 for standard cursors and 8192 for lookahead cursors.

If you use the byte option, the byte maximum is 65535 bytes and the byte minimum specifies the number of bytes contained in a single record. For joins, you must specify the number of bytes contained in two records.

If you use the row option, the row maximum equals the maximum number of records that can be fit in 65535 bytes. The row minimum is 1 row for a single table and 1 rows for a join.

The default is 30000.

{ DEBUG **{** SQL **|** EXTENDED *diag-option* **}** **}** **|** NO-DEBUG

Specifies whether the DataServer should print debugging information for the query to the `dataserv.lg` file.

The SQL option prints the SQL executed by the DataServer against the non-OpenEdge DBMS. The extended option prints additional information, such as cursor statistics. The information you get when you use the EXTENDED option can be helpful in setting your parameters.

The default is NO-DEBUG.

EXTENDED *diag-option*

The syntax for the diagnostic options is as follows:

```
EXTENDED CURSOR | DATA-BIND | PERFORMANCE | VERBOSE
```

For more information, see the OpenEdge DataServer Guides (*Use the Microsoft SQL Data Server* and *Use the Oracle Data Server*).

HINT

This option is only available for the DataServer for ORACLE.

Specifies the ORACLE hint syntax that the DataServer passes directly to the ORACLE DBMS as part of the query. This allows you to control which hints are passed as opposed to the index hints that the DataServer passes when appropriate.

INDEX-HINT **|** NO-INDEX-HINT

This option is available only for the DataServer for ORACLE.

Specifies whether the DataServer provides index hints to the ORACLE DBMS. INDEX-HINT places index hints in the generated SQL; NOINDEX-HINT prevents the use of index hints.

The default is INDEX-HINT.

JOIN-BY-SQLDB **|** NO-JOIN-BY-SQLDB

Specifies whether the non-OpenEdge DBMS can perform joins when possible, which usually improves performance.

The default is JOIN-BY-SQLDB.

LOOKAHEAD **|** NO-LOOKAHEAD

Specifies whether the DataServer uses lookahead or standard cursors. Lookahead cursors fetch as many records as can fit into the allocated cache, which reduces the number of database accesses and improves performance.

The default is LOOKAHEAD, except with statements that use an EXCLUSIVE lock.

ORDERED-JOIN

Specifies that the DataServer embed the ORDERED hint syntax in the SQL it generates. Applies to ORACLE only.

`REVERSE-FROM`

Specifies that tables are joined in the reverse order in which they appear in the FROM clause. Applies to ORACLE only.

`SEPARATE-CONNECTION | NO-SEPARATE-CONNECTION`

Creates a new connection for each cursor that the DataServer opens. Applies to the OpenEdge DataServer for Microsoft SQL Server only.

Example

The following code fragment illustrates a QUERY-TUNING phrase in a FOR EACH statement. In this example, the DataServer uses lookahead cursors with a cache size of 32K and records debugging information:

```
FOR EACH Customer, EACH Order OF Customer WHERE Order.OrdNum > 20
  BY Customer.CustNum
  QUERY-TUNING(LOOKAHEAD CACHE-SIZE 32768 DEBUG EXTENDED) TRANSACTION:
```

Note

For the DataServer for ORACLE, all options of the QUERY-TUNING phrase are effective at both compile and run time, except INDEX-HINT, NO-INDEX-HINT, JOIN-BY-SQLDB, and NO-JOIN-BY-SQLDB, which are only effective at compile time.

For more information on the QUERY-TUNING phrase, see the OpenEdge DataServer Guides (*Use the Microsoft SQL Data Server* and *Use the Oracle Data Server*).

See also

DO statement, FOR statement, OPEN QUERY statement, REPEAT statement

QUIT statement

Raises the `QUIT` condition. By default, this exits from the ABL session and returns to the operating system.

When `QUIT` is executed from within a procedure running on an application server, it terminates the request and closes the connection to the client that initiated the request.

When `QUIT` is executed from within a procedure running on a Progress Application Server for OpenEdge (PAS for OpenEdge), the ABL session is reset to its initial state, which includes deletion of persistent procedures and static ABL objects, the disconnection of databases (or re-connection if the databases were connected at startup), and the clean-up of all global data, such as shared variables. Control then returns to the ABL client session from which the server session was accessed, and the server session is returned to its PAS for OpenEdge session pool for access by other client requests.

Note: Does not apply to SpeedScript programming.

Syntax

```
QUIT
```

Example

This procedure displays a menu. If you choose the last menu item, Exit ABL, the procedure processes the QUIT statement.

r-quit1.p

```
DEFINE SUB-MENU cusmaint1
  MENU-ITEM crecust LABEL "Create New Customer"
  MENU-ITEM chgcust LABEL "Chan&ge Existing Customer"
  MENU-ITEM delcust LABEL "Delete Customer"
  MENU-ITEM prtcust LABEL "Print Customer List"
  MENU-ITEM extcust LABEL "E&xit ABL".

DEFINE MENU mainbar MENUBAR
  SUB-MENU cusmaint1 LABEL "Customer".

ON CHOOSE OF MENU-ITEM crecust
  RUN newcust.p.

ON CHOOSE OF MENU-ITEM chgcust
  RUN chgcust.p.

ON CHOOSE OF MENU-ITEM delcust
  RUN delcust.p.

ON CHOOSE OF MENU-ITEM prtcust
  RUN prncust.p.

ON CHOOSE OF MENU-ITEM extcust
  QUIT.

ASSIGN
  CURRENT-WINDOW:MENUBAR = MENU mainbar:HANDLE
  CURRENT-WINDOW:VISIBLE = TRUE.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- To modify the QUIT statement, add the ON QUIT phrase to a block.
- If QUIT is executed during a transaction, the AVM commits the transaction before exiting.
- In a PAS for OpenEdge application, you typically use this statement for a client connected using the session-managed application model. In this case, you execute QUIT as the final statement in the PAS for OpenEdge Disconnect procedure after setting the SERVER-CONNECTION-BOUND-REQUEST attribute on the SESSION handle to FALSE. This cleans up the server session for access by the next client request.

See also

ON QUIT phrase, STOP statement

QUOTER function

Converts the specified data type to CHARACTER and encloses the results in quotes when necessary.

The QUOTER function is intended for use in QUERY-PREPARE where a character predicate must be created from a concatenated list of string variables to form a WHERE clause. In order to process variables, screen values, and input values so that they are suitable for a query WHERE clause, it is often necessary to enclose them in quotes. For example, European-format decimals and character variables must always be enclosed in quotes. You can use the QUOTER function to meet that requirement.

Syntax

```
QUOTER ( expression [ , quote-char [ , null-string ] ] )
```

expression

An *expression* in the data type that you want to convert to character and enclose with quotes.

quote-char

Either a single or double quote, enclosed in the opposite: "'" or '"'. The default is double quote. Passing ? for this argument results in double quotes.

null-string

The string you want for an unknown value: the word NULL or "" for example. The default is an unquoted question mark, which is the Unknown value (?).

For example, the following code:

```
DEFINE VARIABLE mychar As CHARACTER NO-UNDO INITIAL "Lift Line Skiing".
...
qhandle:QUERY-PREPARE("FOR EACH Customer WHERE Customer.Name = " +
  QUOTER(mychar))
```

Would produce this prepare-string:

```
FOR EACH Customer WHERE Customer.Name = "Lift Line Skiing".
```

Notes

- To address the situation where an Unknown value (?) in a list of concatenated strings could cause the entire string to be unknown and the QUERY-PREPARE to fail, the QUOTER function does **not** return the Unknown value (?) if the *expression* argument is unknown. Instead, it returns a known character value consisting of an UNQUOTED question-mark, by default, or the 3rd argument, if it is present.
- Also, in this situation, a quoted question-mark is not used because it is interpreted as string data in a WHERE clause. After the concatenation is complete, the AVM supplies a normal question mark.

  For example, the following code:

```
DEFINE VARIABLE mychar As CHARACTER NO-UNDO.
...
mychar = ?.
qhandle:QUERY-PREPARE("FOR EACH Customer WHERE Customer.Name = " +
  QUOTER(mychar)).
```

Would produce this prepare-string:

```
FOR EACH Customer WHERE Customer.Name = ?.
```

However, giving the 3rd parameter as `"NULL"` produces `NULL` rather than ?.

- For noncharacter data types, if *expression* is of type DECIMAL, INT64, INTEGER, DATE, DATETIME, DATETIME-TZ, and so on, the following occurs:
  - The *expression* is converted to character and enclosed in quotes. The conversion is similar to the EXPORT format. DATE types, however, always have the 4-digit year.
  - Data types with no DISPLAY format like MEMPTR and LVARBINARY return the Unknown value (?).
  - If a data type is of type RAW, it is converted to base 64.

  For example, the following code:

```
DEFINE VARIABLE mydec As DECIMAL NO-UNDO INITIAL 12.34.
...
qhandle:QUERY-PREPARE("FOR EACH Customer WHERE Customer.Balance = " +
  QUOTER(mydec)).
```

  Would produce this prepare-string:

```
FOR EACH Customer WHERE Customer.Balance = "12.34".
```

  This is especially important for European format decimals that look like 12,34 and would not compile in the above statement unless they are enclosed in quotes.

- If *expression* is of data type CHARACTER, internal quotes are doubled. If the first and last byte are already quotes, then it is assumed that the quoting has already been done, and no further quotes are applied.
- You can use the QUOTER function with an object reference for a class instance to obtain a unique object identifier within the session as a quoted character string.
- When applied to an enum instance, QUOTER will return the enumeration member name as a string. If the enum instance is a flag enum with multiple flags set, the string will be a comma-delimited list of the flags.

R-INDEX function

Returns an INTEGER value that indicates the position of the target string within the source string. In contrast to the INDEX function, R-INDEX performs the search from right to left.

Syntax

```
R-INDEX ( source , target [ , starting ] )
```

source

A constant, field name, variable name, or expression that results in a CHARACTER or LONGCHAR value.

target

A CHARACTER or LONGCHAR expression whose position you want to locate in *source*. If *target* does not exist within *source*, R-INDEX returns 0.

If a *starting* parameter is not specified, then the search for the *target* pattern begins at the right-most character. Even though the search is started from the right, the *target* position is calculated from the left. For example, this code returns a 3 rather than a 2:

```
R-INDEX("abcd" , "c")
```

starting

An integer that specifies the begin point for the search. The search is right-to-left and starts from the starting point. For example, this statement returns 1 R-INDEX("abcdefabcdef","abc",6).

Examples

This procedure prompts you to enter a character string and a pattern to match against the string. It then displays the starting position of the string where the pattern was found.

r-rindex.p

```
DEFINE VARIABLE rindx  AS INTEGER   NO-UNDO.
DEFINE VARIABLE source AS CHARACTER NO-UNDO FORMAT "X(45)".
DEFINE VARIABLE target AS CHARACTER NO-UNDO FORMAT "X(45)".

REPEAT:
  PROMPT-FOR source LABEL "Enter a character string to do pattern matching:"
    WITH FRAME s1 CENTERED.
  PROMPT-FOR target LABEL "Enter a pattern to match in the string:"
    WITH FRAME t1 CENTERED.
  rindx = R-INDEX(INPUT source, INPUT target).
  IF rindx < > 0 THEN DO:
    DISPLAY "The target pattern:" INPUT target NO-LABEL
      "last appears in position" rindx NO-LABEL SKIP
      WITH FRAME r1 ROW 12 CENTERED.
    DISPLAY "in the source string:" INPUT source NO-LABEL
      WITH FRAME r1 ROW 12 CENTERED.
    HIDE FRAME r1.
  END.
  IF rindx = 0 THEN DO:
    DISPLAY "The target pattern:" INPUT target NO-LABEL
      "could not be found" SKIP
      WITH FRAME r2 ROW 12 CENTERED.
    DISPLAY "in the source string:" INPUT source NO-LABEL
      WITH FRAME r2 ROW 12 CENTERED.
    HIDE FRAME r2.
  END.
END.
```

This example also uses a *starting* value:

r-rndex.p

```
DEFINE VARIABLE mark       AS INTEGER   NO-UNDO.
DEFINE VARIABLE line-width AS INTEGER   NO-UNDO.
DEFINE VARIABLE paragraph  AS CHARACTER NO-UNDO.

paragraph = "The course centers around an existing small "
          + "application that you modify to improve perfo"
          + "rmance. Our highly-qualified instructors dem"
          + "onstrate proven analysis and coding techniqu"
          + "es and provide tips for making the most of y"
          + "our ABL code. You are encouraged to bri"
          + "ng your own application problems to class an"
          + "d actively participate in class discussions "
          + "and hands-on lab exercises.".

SET line-width LABEL "Justify with how many characters wide?"
  VALIDATE(line-width >= 20 AND line-width <= 70,
    "Must be between 20 and 70 for this example.")
  WITH SIDE-LABELS FRAME ask.

FORM paragraph FORMAT "x(72)"
  WITH DOWN NO-LABELS USE-TEXT.

DISPLAY "L" + FILL("-", line-width - 2) + "R" @ paragraph.
DOWN.

DO WHILE LENGTH(paragraph) > line-width:
  mark = R-INDEX(paragraph, " ", line-width).
  DISPLAY SUBSTRING(paragraph, 1, mark) @ paragraph.
  DOWN.
  paragraph = SUBSTRING(paragraph, mark + 1).
END.

IF paragraph <> "" THEN
  DISPLAY paragraph.
```

Notes

- If either operand is case sensitive, then the R-INDEX function is also case sensitive.
- If either the *source* string or *target* pattern is null, the result is 0.
- The R-INDEX function is double-byte enabled. You can specify *target* and *source* strings for the R-INDEX function that contain double-byte characters.

See also

INDEX function, LOOKUP function

RADIO-SET phrase

Describes a radio set representation for a field or variable. The RADIO-SET phrase is an option of the VIEW-AS phrase.

Note: Does not apply to SpeedScript programming.

Syntax

```
RADIO-SET
  [ HORIZONTAL [ EXPAND ] | VERTICAL ]
  [ size-phrase ]
  RADIO-BUTTONS label, value [ , label , value ] . . .
  [ TOOLTIP tooltip ]
```

HORIZONTAL

Specifies that the radio buttons are aligned horizontally. Vertical alignment is the default.

VERTICAL

Specifies that the radio buttons are aligned vertically. Because this is the default alignment, you do not have to supply this attribute.

EXPAND

Pads all button labels to be the width of the widest radio button label. This ensures that the buttons are evenly spaced. Use this option only in conjunction with the HORIZONTAL option. If you do not specify this option, the individual radio buttons are spaced evenly if the lengths of the labels vary.

size-phrase

Specifies the outside dimensions of the radio-set widget. This is the syntax for *size-phrase*:

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

For further information, see the SIZE phrase reference entry.

RADIO-BUTTONS *label, value* [, *label, value*] . . .

A list of radio buttons whose selections are mutually exclusive. Each button is composed of a label and value pair. The *label* is a character string that is the label for the radio button. The *value* is the value to be assigned to the field or variable if the radio button is selected; *value* must be a valid value for the field or variable.

You can designate a character within each *label* as a navigation mnemonic in Windows. Indicate the character by preceding it with an ampersand (&). When the radio set is displayed, the mnemonic is underlined. The user can choose to the specific button by pressing **ALT** and the underlined letter.

Note: If two or more buttons of a radio set use the same label, the AVM uses only the value of the first button.

TOOLTIP *tooltip*

Allows you to define a help text message for a text field or text variable. The AVM automatically displays this text when the user pauses the mouse button over a text field or text variable for which a tooltip is defined.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the tooltip is removed. No tooltip is the default. The TOOLTIP option is supported in Windows only.

Example

This procedure displays a radio set that consists of three radio buttons and prompts the user to select one of the buttons. When the user selects the button, the program displays the text "This event occurred on" and the date value of selected button.

r-radio1.p

```
DEFINE VARIABLE hist-date AS DATE NO-UNDO
  FORMAT "99/99/9999" INITIAL 07/04/1776
  VIEW-AS RADIO-SET RADIO-BUTTONS
    "Declaration of Independence", 07/04/1776,
    "Lee Surrenders to Grant", 04/07/1865,
    "Man Walks on Moon", 07/11/1969.

FORM hist-date
  WITH FRAME main-frame NO-LABELS TITLE "Dates in US History".

ON VALUE-CHANGED OF hist-date DO:
  ASSIGN hist-date.
  DISPLAY "This event occurred on " + STRING(hist-date) FORMAT "x(60)"
    WITH FRAME main-frame.
END.

ENABLE hist-date WITH FRAME main-frame.APPLY "VALUE-CHANGED" TO hist-date.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

See also

VIEW-AS phrase

RANDOM function

Returns a random INTEGER value between two integers (inclusive).

Note: This function returns a number from a **pseudorandom** sequence of numbers rather than a truly random sequence.

The Alternate Random Number Generator (-rand) parameter determines whether the same sequence of random numbers is generated for each session. For information on this parameter, see *Startup Command and Parameter Reference*.

Syntax

```
RANDOM ( low , high )
```

low

An integer expression that is the lower of the two expressions you are supplying to the RANDOM function. This can be an INT64 value.

high

An integer expression that is the higher of the two expressions you are supplying to the RANDOM function. This can be an INT64 value.

Example

Often when you set up a database for testing purposes, you want to generate many records without actually keying in data for each record. The `r-random.p` procedure generates 10 Order records and a random number of OrderLines for each Order record.

r-random.p

```
DEFINE VARIABLE onum  AS INTEGER NO-UNDO.
DEFINE VARIABLE olnum AS INTEGER NO-UNDO.

DO onum = 1 TO 10 TRANSACTION:
  CREATE Order.
  ASSIGN
    Order.OrderNum  = onum
    Order.OrderDate = TODAY.

  DO olnum = 1 TO RANDOM(1,9):
    CREATE OrderLine.
    ASSIGN
      OrderLine.LineNum = olnum
      OrderLine.ItemNum = olnum.
  END.
END.
```

RAW function
(ORACLE only)

Extracts bytes from a field.

Syntax

```
RAW ( field [ , position [ , length ] ] )
```

field

Any field from which you want to extract bytes.

position

An integer expression that indicates the position of the first byte you want to extract from *field*. The default value of *position* is 1.

length

An integer expression that indicates the number of bytes you want to extract from *field*. If you do not use the *length* argument, RAW uses *field* from *position* to end.

Example

This procedure extracts bytes from the Name field of the first Customer, starting at byte 8, and writes 4 bytes to the variable r1:

r-rawfct.p

```
/*You must connect to a non-OpenEdge demo database to run this procedure*/
DEFINE VARIABLE r1 AS RAW NO-UNDO.

FIND FIRST Customer NO-LOCK.
r1 = RAW(Customer.Name,8,4).
```

Notes

- If *position* is less than 1, or *length* is less than 0, the AVM returns a run-time error.
- If (*position* + *length* -1) is greater than the length of the field from which you are extracting the bytes, the AVM returns a run-time error.

See also

GET-BYTE function, LENGTH statement, PUT-BYTE statement, RAW statement

---

RAW statement
(ORACLE only)

Writes bytes to a field.

Syntax

```
RAW ( field [ , position [ , length ] ] ) = expression
```

field

The field in which you want to store *expression*.

position

An integer expression that indicates the position in *field* where you want to store *expression*. The default for *position* is 1.

length

An integer expression that indicates the number of positions you want to replace in *field*. If you do not use the *length* argument, RAW puts *expression* into *field* from *position* to end.

ABL treats variable-length fields and fixed-length fields differently. See the Notes section for more information.

expression

A function or variable name that returns data and results in the bytes that you want to store in *field*.

Notes

- In a variable length field, if (*position* + *length* -1) is greater than the length of *field*, the AVM pads the field with nulls before it performs the replacement.
- In a fixed length field, if (*position* + *length* -1) is greater than the length of *field*, the AVM returns a run-time error. If (*position* + *length* -1) is less then the length of *field*, the AVM pads the field with nulls so that it remains the same size.
- If *position*, *length*, or *expression* is equal to the Unknown value (?), then *field* becomes the Unknown value (?).
- If *position* is less than 1, or *length* is less than 0, the AVM generates a run-time error.

See also

GET-BYTE function, LENGTH function, LENGTH statement, PUT-BYTE statement, RAW function

---

RAW-TRANSFER statement

Copies a record wholesale from a source to a target.

Syntax

```
RAW-TRANSFER
  { [ BUFFER ] buffer    TO [ FIELD ] raw-field
  | [ FIELD ] raw-field  TO [ BUFFER ] buffer
  | [ BUFFER ] buffer    TO [ BUFFER ] buffer
  }
[ NO-ERROR ]
```

BUFFER

Specifies a parameter is a buffer.

buffer

A source or target database record.

Note: If the source buffer contains only a partial field list, RAW-TRANSFER fails.

FIELD

Specifies a parameter is a raw-field.

raw-field

A source or target data field of type RAW.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

The following ABL example performs a RAW-TRANSFER of a newly created Customer record to the Record field of Replication-Log table:

```
TRIGGER PROCEDURE FOR REPLICATION-CREATE OF Customer.
CREATE Replication-Log.
ASSIGN
  Replication-Log.Taskid = DBTASKID(LDBNAME(BUFFER Replication-Log))
  Replication-Log.Table  = 'Customer'
  Replication-Log.Action = 'CREATE'.
RAW-TRANSFER Customer TO Replication-Log.Record.
```

For more information on database replication, see *Manage the OpenEdge Database*.

Notes

- The RAW-TRANSFER statement has several variations:
  - The "buffer to raw-field" variation copies the entire record from the buffer to the raw field, prepending information on the source schema to the raw field.
  - The "raw-field to buffer" variation first checks that the source schema information prepended to the raw field matches the schema of the buffer. Then it creates a target record, if necessary. Finally it updates each key field in the new record using values from the raw field, which forces indexing to occur.
  - The "buffer to buffer" variation is the same as the "raw-field to buffer" variation, except that the source is a record in another buffer.
- The RAW-TRANSFER statement respects database triggers.
- You can marshal an OpenEdge database record so that it can be sent across sockets by using the RAW-TRANSFER statement to put the record into a RAW variable and then copying the RAW variable to a MEMPTR that is being written to a socket. Use the PUT-BYTES function to do this. You can unmarshal database records by using the GET-BYTES function and then RAW-TRANSFER.
- At run time, the RAW-TRANSFER statement:
  - Checks that the signatures of the source data and the target data match.
  - Compares source and target code page ids, and (if they are present and different) translates the source's character data, writing any warnings to the database log file and raising any error conditions.
  - Creates the target record, if none exists, and runs all appropriate CREATE and REPLICATE-CREATE triggers (unless the DISABLE TRIGGERS FOR LOAD option is active for the target).
  - Registers changes in key fields with the index manager by updating each key field in the target when it differs from the source.
  - Copies all data from the source record to the target record.

- Executes ASSIGN triggers for any modified fields (unless the DISABLE TRIGGERS FOR LOAD option is active for the target).

- When using the RAW-TRANSFER statement to copy a record that contains a BLOB or CLOB field, the AVM skips the BLOB or CLOB field and stores the Unknown value (?) in the BLOB or CLOB field of the target record.

See also

DISABLE TRIGGERS statement, LDBNAME function, NO-ERROR option, RAW-TRANSFER( ) method, RECORD-LENGTH function

READKEY statement

Reads one keystroke from an input source and sets the value of LASTKEY to the keycode of that keystroke. Use the READKEY statement when you want to look at each keystroke a user makes and take some action based on that keystroke.

Note: Does not apply to SpeedScript programming.

Syntax

```
READKEY [ STREAM stream | STREAM-HANDLE handle ][ PAUSE n ]
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream. See the DEFINE STREAM statement.

STREAM-HANDLE *handle*

Specifies the handle to a stream. If *handle* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

PAUSE *n*

The READKEY statement waits up to *n* seconds for a keystroke. You can choose a whole or a fractional value for the time-out interval. If the time-out period you specify is a fractional value, the value is rounded to the nearest whole millisecond. If you do not press a key during that amount of time, READKEY ends, and sets the value in LASTKEY to -1.

PAUSE 0 causes READKEY to immediately return a value. If no character is available, READKEY sets the value of LASTKEY to -1. Use this form of READKEY to do polling through UNIX pipes or terminal ports.

Example

In the following procedure, when the user presses a key, the READKEY statement reads the keystroke and stores the character code value of that key (the key code) as the value of LASTKEY. The CHR function converts the character code value into a character value. If the character value is a Y, the AVM deletes the Customer. KEYFUNCTION determines the function of the LASTKEY. If that function is END-ERROR, the AVM exits the block, ending the procedure.

r-readky.p

```
FOR EACH Customer:
  DISPLAY Customer.CustNum Customer.Name Customer.Address Customer.City
    Customer.State WITH 1 DOWN.
  MESSAGE "If you want to delete this customer, press Y".
  MESSAGE "Otherwise, press any other key.".
  READKEY.
  IF CHR(LASTKEY) = "Y" THEN DELETE Customer.
  ELSE IF KEYFUNCTION(LASTKEY) = "END-ERROR" THEN LEAVE.
END.
```

Notes

- If you use READKEY, it intercepts any input from the user. Thus no widgets receive the input. To pass the input to a widget, you must use the APPLY statement.
- The READKEY function is double-byte enabled. The READKEY function returns values only after the input method places the data in the keyboard buffer. It returns the key code of the most recent key sequence returned from the keyboard buffer. A key sequence is the set of keystrokes necessary to generate one character or function key event in ABL.
- If the current input source is a file, then READKEY reads the next character from that file and returns the value of that character (1 to 255) to LASTKEY. READKEY does not translate periods (.) in the file into the ENDKEY value. It does translate end of line into RETURN (13), but it cannot read any special keys, such as function keys.

  When the AVM reaches the end of the file, it sets the value of LASTKEY to -2, but does not close the input file. At that point, an APPLY LASTKEY (same as APPLY -2) raises the ENDKEY condition.
- If the current input source is a UNIX pipe, any timer you set with the PAUSE option might expire before READKEY can read a character. If so, LASTKEY is set to -1.
- If the last key typed is an invalid character sequence, READKEY sets the value of LASTKEY to -1.
- READKEY is included in the set of statements that are considered input statements. If a block contains an input statement, the AVM assumes that the next block iteration may be working with different data and therefore the AVM does not do infinite loop protection. Therefore, having READKEY in a block will allow UNDO, RETRY to take effect, whereas without it (or any other input statement), it would be changed to UNDO, LEAVE in order to avoid an infinite loop.
- For more information on monitoring keystrokes, see *OpenEdge Programming Interfaces*.
- In the context of the .NET blocking method, `System.Windows.Forms.Application:Run( )`, if you execute the READKEY statement for a non-modal ABL window from within a user-defined function or non-VOID method, this statement raises the STOP condition. For more information on the .NET `Application:Run( )` method, see the reference entry for the WAIT-FOR statement (.NET and ABL). For more information on editing blocks, see the EDITING phrase entry.

See also

DEFINE STREAM statement, EDITING phrase, LASTKEY function, Stream object handle

RECID function

Returns the unique internal identifier of the database record currently associated with the record buffer you name. This internal identifier has the data type RECID, a four-byte value that is supported by OpenEdge databases and some non-OpenEdge DataServers.

This function is supported for backward compatibility. For most applications, use the ROWID function, instead. For more information, see the ROWID function reference entry.

Syntax

```
RECID ( record )
```

record

The name of the record whose RECID you want.

To use the RECID function with a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

Example

You might decide that you do not want to lock a record until the user starts to update that record. In the example procedure, the FIND statement reads a Customer record without locking the record. The RECID function puts the internal database identifier of that record in the `crecid` variable. If the user decides to update the `CreditLimit` field, the procedure finds the record again using the value in `crecid`. The second FIND statement reads the record again, this time placing an EXCLUSIVE-LOCK on it. Because the record is first found with NO-LOCK, it is possible for the record to be updated by another user after the first FIND and before the second.

r-recid.p

```
DEFINE VARIABLE response AS LOGICAL NO-UNDO.
DEFINE VARIABLE crecid   AS RECID   NO-UNDO.

REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer NO-LOCK USING Customer.CustNum.
  crecid = RECID(Customer).
  DISPLAY Customer.name.
  response = YES.
  UPDATE response LABEL "Update credit-limit ?".
  IF response THEN DO:
    FIND Customer WHERE RECID(Customer) = crecid EXCLUSIVE-LOCK.
    UPDATE Customer.CreditLimit.
  END.
END.
```

Notes

- Use the RECID function to rapidly retrieve a previously identified record, even if that record has no unique index.

- If you want a called procedure to use the same record as a calling procedure, use the RECID function to ensure that you are retrieving the same record. Use a SHARED variable to communicate the RECID of a record from one procedure to another. The second procedure can then find the same record. This is an alternative to using shared buffers.
- Avoid storing RECID values in database fields because those RECIDs will change if you dump and reload the database.
- You cannot use a RECID to identify the partition of a record in a partitioned table. Using a stored RECID to retrieve a record in a partitioned table will not work unless the record is in the initial composite partition. Use ROWID instead.
- You do not have to explicitly check to see whether a record is AVAILABLE before using the RECID function. The RECID function returns the Unknown value (?) if a record cannot be accessed.

  This example displays a RECID only when a record can be accessed:

```
DISPLAY (IF AVAILABLE Customer THEN RECID(Customer) ELSE ?).
```

  Directly reference RECID even if a record cannot be found:

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.CustNum.
END.
DISPLAY RECID(Customer).
```

See also

DEFINE BUFFER statement, DEFINE VARIABLE statement, Record phrase, ROWID function, VAR statement

Record phrase

Identifies the record or records you want to verify using the CAN-FIND function, retrieve with a FIND statement, query with a FOR statement or OPEN QUERY statement, or preselect in a DO or REPEAT block.

The Record phrase syntax describes three kinds of information:

- Qualifies the record(s) to access in the table
- Specifies the index to use when locating records
- Defines the type of record lock to apply when the records are read

Syntax

```
{ record [ field-list ] }
[ constant ]
[ [ LEFT ] OUTER-JOIN ]
[ OF table ]
[ WHERE expression ]
[ TENANT-WHERE expression [ SKIP-GROUP-DUPLICATES ] ]
[ USE-INDEX index | TABLE-SCAN ]
[ USING [ FRAME frame ] field
  [ AND [ FRAME frame ] field ] . . . ]
[ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ]
[ NO-PREFETCH ]
```

Note: You can specify the OUTER-JOIN, OF, WHERE, USE-INDEX, TABLE-SCAN, and USING options in any order. You cannot use *field-list* in an OPEN QUERY statement. You cannot use OUTER-JOIN or EXCLUSIVE-LOCK in a CAN-FIND function.

record

The name of a database table, a temp-table that you named in a DEFINE TEMP-TABLE statement, or a buffer that you named in a DEFINE BUFFER statement. If *record* is the name of a table, *record* actually refers to the default buffer of the specified database table or temp-table. Also, the default buffer for a database table is always scoped to the main external procedure or class definition block even if you reference the default buffer in a sub-block, such as an internal procedure or method of a class.

To access a record in a table defined for multiple databases, you must qualify the record's table name with the database name. Use this syntax to refer to a record in a table for a specific database:

```
dbname.tablename
```

You do not have to qualify the reference if *record* is the name of a defined buffer.

field-list

Specifies a list of fields to include or exclude when you retrieve records.

Field lists allow you to retrieve only specified fields rather than all the fields of a record. They can improve performance over a network by enabling more records to fit into a single network message. Field lists can be added to a FOR, DO PRESELECT, or REPEAT PRESELECT statement. They are also available for queries using the DEFINE QUERY statement, except when the query is against a temp-table. The following is the syntax for *field-list*:

```
{    FIELDS [ ( [field . . .] ) ]
   | EXCEPT [ ( [field . . .] ) ]
}
```

The FIELDS option specifies the fields you want to include in a record retrieval, and the EXCEPT option specifies the fields that you want to exclude from a record retrieval. The *field* parameter is the name of a single field in the specified table. If *field* is an array reference, the whole array is retrieved. Specifying FIELDS with no *field* references retrieves no fields. However the ROWID value for a specified record is available using the ROWID function. Specifying EXCEPT with no *field* references or specifying *record* without a *field-list* causes the AVM to retrieve a complete record.

Note:

Field lists should be used with caution because they can cause unexpected run-time errors. For example, it is possible that you did not include a field that is required by a new or revised subroutine or trigger. The result can be a run-time error that may be difficult to debug. A workaround is to use either the `-fldisable` or the `-rereadfields` client session startup parameter. This will allow your application to run (although more slowly) until the application can be fixed.

Note the difference between `-fldisable` and `-rereadfields`. The Field List Disable (`-fldisable`) parameter causes the AVM to ignore **all** field lists and to fetch the entire record for **every** query. The Reread Fields (`-rereadfields`) parameter causes the AVM to fetch the record with the specified fields, but then re-fetch the entire record if an ABL statement tries to reference a field that is not there. Also note that when the re-fetch happens, any `FIND` trigger is not run again. For more information on these startup parameters, see *Startup Command and Parameter Reference*.

This statement retrieves only the `Name` and `Balance` fields of the Customer table:

```
FOR EACH customer FIELDS (name balance): DISPLAY name balance.
```

This statement retrieves all fields of the Customer table except the `Name` and `Balance` fields:

```
FOR EACH Customer EXCEPT (Name Balance):
  DISPLAY Customer EXCEPT Customer.Name Customer.Balance.
```

When you specify a field list, the AVM might retrieve additional fields or the complete record depending on the type of retrieval operation and the DataServer that provides the record. Thus, the AVM:

- Retrieves any additional fields required by the client to complete the record selection.
- Retrieves a complete record when the record is fetched with EXCLUSIVE-LOCK. This ensures proper operation of updates and the local before-image (BI) file. For information on the local BI file, see *Manage the OpenEdge Database*.
- Retrieves a complete record for DataServers that do not support SHARE-LOCK. For more information, see the appropriate DataServer guides (*Use the Microsoft SQL Data Server* and *Use the Oracle Data Server*).

Note: Always specify fields that you plan to reference in the field list. Only those extra fields that the client requires for record selection are added to the specified field list. The AVM distributes record selection between the client and server depending on a number of factors that change with each OpenEdge release. Therefore, never rely on fields that you did not specify but which the AVM fetches for its own needs; they might not always be available. There is no additional cost to specify a field in the list that you otherwise expect the AVM to provide.

This statement retrieves the Customer.CustNum field in addition to those specified in the field lists because it is required to satisfy the inner join between the Customer and Order tables:

```
FOR EACH Customer FIELDS(Name) NO-LOCK,
 EACH Order FIELDS(OrderNum SalesRep) OF Customer NO-LOCK:
 DISPLAY Customer.Name Customer.CustNum Order.OrderNum Order.SalesRep.
```

However, do not rely on the AVM to always provide such extra fields. For reliability, add the CustNum field to the Customer field list as follows:

```
FOR EACH Customer FIELDS(Name CustNum) NO-LOCK,
 EACH Order FIELDS(OrderNum SalesRep) OF Customer NO-LOCK:
 DISPLAY Customer.Name Customer.CustNum Order.OrderNum Order.SalesRep.
```

constant

The value of a single component, unique, primary index for the record you want. This option is not supported for the OPEN QUERY statement:

```
FIND Customer 1.
```

The AVM converts this FIND statement with the *constant* option of 1 to the following statement:

```
FIND Customer NO-LOCK WHERE Customer.CustNum = 1.
```

The CustNum field is the only component of the primary index of the Customer table.

If you use the *constant* option, you can use it only once in a single Record phrase, and it must precede any other options in the Record phrase.

[LEFT **]** OUTER-JOIN

Specifies a left outer join between *record* and the table (or join) specified by the previous Record phrase(s) of an OPEN QUERY statement. A left outer join combines and returns data from the specified tables in two ways. First, the records selected for the table (or join) on the left side combine with each record selected using the OF or WHERE options from the table on the right (*record*). Second, the records selected for the table (or join) on the left side combine with the Unknown value (?) for the fields from the table on the right (*record*) for which no records are selected using the OF or WHERE options. The join is ordered according to the given sort criteria starting with the left-most table in the query.

Note: If you specify the OUTER-JOIN option, you must also specify the OUTER-JOIN option in all **succeeding** Record phrases of the query to obtain a left outer join. That is, for multiple Record phrases, all joins in the query following your first left outer join must also be left outer joins. Otherwise, the result is an inner join for all records up to the last inner join in the query. For more information, see *ABL Essentials*.

The OUTER-JOIN option is supported only in the OPEN QUERY statement and in Record phrases specified after the first Record phrase in the OPEN QUERY statement. The LEFT keyword is optional with OUTER-JOIN. If you specify OUTER-JOIN, you must also specify the OF option, WHERE option, or any combination of the OF and WHERE options. These options are required to select *record* (the right-most table) for the specified left outer join. For example:

```
OPEN QUERY q1 PRESELECT EACH Customer,
  FIRST Order OUTER-JOIN OF Customer WHERE Order.OrderNum < 50
  FIRST OrderLine OUTER-JOIN OF Order WHERE OrderLine.ItemNum < 15.
```

This query specifies a left outer join between Customer and Order, and also between that join and OrderLine. Thus, for each Customer record that has no Orders or has no Orders with an `OrderNum` less than 50, the query returns the Customer fields and ? for all fields of the Order and OrderLine tables. In addition, if there are no OrderLine records with `ItemNum` less than 15 for any selected Customer and Order, the query returns ? for all fields of OrderLine. Otherwise, it returns each Customer record along with its first selected Order record and OrderLine record.

In all statements where multiple Record phrases are allowed (including DO, FOR, OPEN QUERY, and REPEAT statements), the default join (without the OUTER-JOIN option) is an inner join between *record* and the table (or join) specified by the previous Record phrase(s). An inner join returns the records selected for the table (or join) on the left side combined with each selected record from the table on the right (*record*). For an inner join, no records are returned for the table (or join) on the left for which no record is selected from the table on the right (*record*).

The following query specifies an inner join between Customer and Order, and also between that join and OrderLine. Thus, this query only returns Customer records that have at least one Order with `OrderNum` less than 50 that also have at least one OrderLine with `ItemNum` less than 15, and it returns just the first such Order and OrderLine for each Customer record.

```
OPEN QUERY q1 PRESELECT EACH Customer,
  FIRST Order OUTER-JOIN OF Customer WHERE Order.OrderNum < 50
  FIRST OrderLine OF Order WHERE OrderLine.ItemNum < 15.
```

Note: If you specify a Record phrase as an inner join, the current Record phrase and all **preceding** Record phrases in the query participate in contiguous inner joins, even if prior Record phrases specify the OUTER-JOIN option. Thus, for multiple Record phrases, all joins in the query up to the right-most inner join result in contiguous inner joins.

For more information on joins in ABL, see *ABL Essentials*.

`OF` *table*

Relates *record* to one other table specified by a table or buffer name (*table*). The relationship is based on common field names between *record* and *table* that also participate in a UNIQUE index for either *record* or *table*. When you use OF and the UNIQUE index is multi-field, all fields in the index participate in the match criteria. A reference to *table* must appear in a prior joined Record phrase in the same statement, or remain in scope from a prior record reading statement, such as a FIND statement.

Note: For the OF keyword to properly detect a relationship between two tables, only one such relationship is allowed.

In this example, the OF option relates the order table to the Customer table; thus the AVM selects the Customer record related to the Order record currently in use. The AVM converts the FIND statement with the OF option to a FIND statement with the WHERE option.

```
PROMPT-FOR Order.OrderNum.
FIND Order NO-LOCK USING Order.OrderNum.
DISPLAY Order.

FIND Customer OF Order NO-LOCK.
DISPLAY customer.
```

You can use WHERE to access related tables, whether or not the field names of the field or fields that relate the tables have the same name. For example:

```
FIND Customer NO-LOCK WHERE Customer.CustNum = Order.CustNum.
```

`WHERE` *`expression`*

Qualifies the records you want to access. The *`expression`* is a constant, field name, variable name, or expression whose value you want to use to select records. You can use the WHERE keyword even if you do not supply an *`expression`*. For example:

```
FOR EACH Customer NO-LOCK WHERE {*}
```

The WHERE clause may not work the same way against a DataServer as it does against the OpenEdge database. Refer to the appropriate DataServer Guide (*Use the Microsoft SQL Data Server* or *Use the Oracle Data Server*), for additional information on how this feature will perform.

Note: You cannot reference an OpenEdge BLOB or CLOB field in a WHERE clause. However, in OpenEdge DataServer for MS SQL Server, a non-legacy BLOB or CLOB server data type can be mapped to the OpenEdge CHARACTER data type in a WHERE clause operand as long as the total size of the resolved WHERE clause (which might include the CHARACTER-mapped value) does not exceed 30000 bytes.

Note: Use the `-noroutineinwhere` parameter to force the compiler to raise an error, if the `WHERE` expression contains a user-defined function, a class method, or a class property, which implements the `GET` method using ABL statements.

In an OPEN QUERY statement or FOR statement, the WHERE clause can use the CONTAINS operator to reference a field with a word index. This is the syntax for the CONTAINS operator:

```
field CONTAINS search-expression
```

In this syntax, *field* represents a field in which a word index has been defined. The *search-expression* specifies one or more words to search for. It must evaluate to a string with this syntax:

```
"word[ [ & | | | ! | ^ ]word]..."
```

Each *word* is a word to search for. The ampersand (&) represents a logical AND; the vertical line (|), exclamation point (!), or caret (^) represent a logical OR. You can use an asterisk (*) as a wildcard, but only at the end of a *word*. Using an asterisk anywhere else in a *word* raises an error.

Here is an example using the CONTAINS clause:

```
FOR EACH Item NO-LOCK WHERE Item.CatDescription CONTAINS "ski":
  DISPLAY Item.ItemName Item.CatDescription VIEW-AS EDITOR SIZE 60
   BY 15.
END.
```

You can use parenthesis to establish precedence in the search string. For example, the following expression returns a subset of the catalog descriptions containing "ski":

```
WHERE Item.CatDescription CONTAINS "ski & (gog* ! pol*)"
```

The CONTAINS operator must be used in an expression on its own, and not in a subexpression (for example, in an IF function).

You cannot apply the NOT operator to a CONTAINS expression.

Note: The CONTAINS option is not allowed in a FIND statement. If the session is started with the Version 6 Query (-v6q) parameter, the CONTAINS option is also not allowed in a FOR statement.

Note: For information about compiling, storing, and applying the UTF-8 word-break rules to a database, see *Internationalize ABL Applications*.

TENANT-WHERE *expression* [SKIP-GROUP-DUPLICATES]

Allows a super tenant to specify which tenants are to be included in a query, where *expression* is a relational operation, or a logical combination of relational operations, that test for a character or integer expressions that correspond to a tenant name or tenant ID, respectively. In order to use the TENANT-WHERE functionality at run time, you must be a super tenant, not a regular tenant.

You can only use the TENANT-WHERE option in the FOR EACH statement or option of a query. If you use a TENANT-WHERE in a join, you can have only one such option at one level of the join.

Note: This option does **not** change the effective tenancy for a super tenant. For more information on effective tenancy, see the entry for the SET-EFFECTIVE-TENANT function.

The SKIP-GROUP-DUPLICATES option allows you to skip duplicate data for a tenant group.

If a query has been coded with a TENANT-WHERE option, and compiled, but is run by a regular tenant, the AVM raises a run-time error.

Note that any use of the TENANT-ID or TENANT-NAME functions in a TENANT-WHERE expression must **not** specify the optional database parameter, as the compiler assumes the database is the same as for the query in which the TENANT-WHERE option is specified.

For more information on this option, and examples, see the sections on multi-tenant ABL in *OpenEdge Programming Interfaces*.

`USE-INDEX` *`index`*

Identifies the index you want to use while selecting records. If you do not use this option, the AVM selects an index to use based on the criteria specified with the WHERE, USING, OF, or *`constant`* options.

`TABLE-SCAN`

You can use the TABLE-SCAN option as an alternative to the USE-INDEX keyword in FOR EACH statements.

TABLE-SCAN provides the most efficient access method when retrieving all rows of a temp-table or a database table in a Type II Storage area. TABLE-SCAN returns these rows without using an index. When the TABLE-SCAN keyword is used, the AVM only accesses the record block instead of both record and index blocks.

```
FOR EACH record . . . [ USE-INDEX index | TABLE-SCAN ]
```

For a table in a Type I storage, the AVM uses the default index to perform the scan instead of TABLE-SCAN. If the QryInfo log entry is set, the AVM logs message if the TABLE-SCAN keyword is used on a Type I Storage area.

Here is an example using the TABLE-SCAN keyword:

```
FOR EACH mytable TABLE-SCAN:
  totalCost = totalCost + mytable.cost
END.
```

Here *`mytable`* is in a Type II Storage area. The AVM uses TABLE-SCAN instead of WHOLE-INDEX to access the record blocks. The AVM accesses both primary index and record blocks, if the WHOLE-INDEX keyword is used.

For more information, see also *Web Paper: ABL Database Triggers and Indexes*.

`USING [ FRAME` *`frame`* `] field [ AND [ FRAME` *`frame`* `] field] . . .`

One or more names of fields for selecting records. You must have previously entered each field you name in this option, usually with a PROMPT-FOR statement. The field must be viewed as a fill-in or text widget.

The USING option translates into an equivalent WHERE option:

```
PROMPT-FOR Customer.CustNum.
FIND Customer NO-LOCK USING Customer.CustNum.
```

This FIND statement is the same as this statement:

```
FIND Customer NO-LOCK WHERE Customer.CustNum = INPUT Customer.CustNum.
```

The `CustNum` field is a non-abbreviated index. However, if the `Name` field is an abbreviated index of the Customer table, the AVM converts the FIND statement with the USING option. For example:

```
PROMPT-FOR Customer.Name.
FIND Customer NO-LOCK USING Customer.Name.
```

The following statement is a result of the previous one:

```
FIND Customer NO-LOCK WHERE Customer.Name BEGINS INPUT Customer.Name.
```

`SHARE-LOCK`

Tells the AVM to put a SHARE-LOCK on records as they are read. Another user can read a record that is share locked, but cannot update it. By default, the AVM puts a SHARE-LOCK on a record when it is read (unless it uses a CAN-FIND function), and automatically puts an EXCLUSIVE-LOCK on a record when it is modified (unless the record is already EXCLUSIVE-LOCKed).

In a CAN-FIND function, NO-LOCK is the default. Also, CAN-FIND cannot use EXCLUSIVE-LOCK.

If you use the SHARE-LOCK option and the AVM tries to read a record that is EXCLUSIVE-LOCKed by another user, the AVM waits to read the record until the EXCLUSIVE-LOCK is released. The AVM displays a message to the user of that procedure, identifying the table that is in use, the user ID of the user, and the tty of the terminal using the table.

If you are using a record from a work table, the AVM disregards the SHARE-LOCK option.

`EXCLUSIVE-LOCK`

Tells the AVM to put an EXCLUSIVE-LOCK on records as they are read. Other users cannot read or update a record that is EXCLUSIVE-LOCKed, except by using the NO-LOCK option. They can access that record only when the EXCLUSIVE-LOCK is released. The AVM automatically puts a SHARE-LOCK on a record when it is read and automatically puts an EXCLUSIVE-LOCK on a record when it is updated.

If a record is read specifying EXCLUSIVE-LOCK, or if a lock is automatically changed to EXCLUSIVE-LOCK by an update, user's read or update will wait if any other user SHARE-LOCKed or EXCLUSIVE-LOCKed the record.

When a procedure tries to use a record that is EXCLUSIVE-LOCKed by another user, the AVM displays a message identifying the table that is in use, the user ID of the user, and the tty of the terminal using the table.

If you are using a record from a work table, the AVM disregards the EXCLUSIVE-LOCK option. Also, CAN-FIND cannot use the EXCLUSIVE-LOCK option.

Specifying EXCLUSIVE-LOCK causes the AVM to retrieve complete records, even when the record is specified with *`field-list`*.

NO-LOCK

Tells the AVM to put no locks on records as they are read, and to read a record even if another user has it EXCLUSIVE-LOCKed.

Another user can read and update a record that is not locked. By default, the AVM puts a SHARE-LOCK on a record when it is read (unless it uses a CAN-FIND function, which defaults to NO-LOCK), and automatically puts an EXCLUSIVE-LOCK on a record when it is updated (unless the record is already EXCLUSIVE-LOCKed). A record that has been read NO-LOCK must be reread before it can be updated.

```
DEFINE VARIABLE rid AS ROWID NO-UNDO.

rid = ROWID(customer).
FIND Customer EXCLUSIVE-LOCK WHERE ROWID(Customer) = rid.
```

If a procedure finds a record and it places it in a buffer using NO-LOCK and you then re-find that record using NO-LOCK, the AVM does not reread the record. Instead, it uses the copy of the record that is already stored in the buffer. To make sure the AVM gets a fresh copy of the record from the database, you must release all NO-LOCK buffers referencing the record or retrieve the record with EXCLUSIVE-LOCK. Note that you can use the Reread Nolock (`-rereadnolock`) startup parameter to change this default behavior.

When you read records with NO-LOCK, you have no guarantee of the overall consistency of those records because another user might be in the process of changing them. When values are assigned to indexed fields for a newly created record or are modified in an existing record, the index is immediately updated to reflect the change. However the copy of the data record in the buffers used by the database server might not be updated until later in the transaction. For example, the following procedure might display a `CustNum` of 0 if another user's active transaction has created a record and assigned a value to the indexed field `CustNum` that is greater than 100:

```
FOR EACH Customer NO-LOCK WHERE Customer.CustNum > 100
  DISPLAY Customer.CustNum.
END.
```

If you are using a record from a work table, the AVM disregards the NO-LOCK option.

NO-PREFETCH

Specifies that only one record is sent across the network at a time. If you specify *`field-list`*, only the specified fields and any additional fields required for record selection are sent. If you do not specify this option, the AVM can send more than one record from the server to the client in each network packet.

Examples

In the `r-recph.p` procedure, there are two Record phrases that make an inner join between the Customer and Order tables.

r-recph.p

```
FOR EACH Customer WHERE Customer.CreditLimit GE 50000,
  EACH Order OF Customer:
  DISPLAY Customer.CustNum Customer.Name Customer.CreditLimit Order.Orderum
    Order.OrderDate Order.Terms.
END.
```

Using these Record phrases, the FOR EACH block reads a Customer record only if it has a `CreditLimit` value greater than 50000 and at least one Order record associated with it.

r-recph2.p

```
REPEAT:
  FIND NEXT Customer USE-INDEX CountryPost WHERE Customer.Name BEGINS "S"
    EXCLUSIVE-LOCK.
  UPDATE Customer.Name Customer.Country Customer.PostalCode Customer.Phone.
END.
```

In the `r-recph2.p` procedure, there is one Record phrase:

```
Customer USE-INDEX CountryPost WHERE Customer.Name BEGINS "S" EXCLUSIVE-LOCK
```

Using the zip index named country-post rather than the `CustNum` index (the primary index for the Customer table), the FIND statement reads only those Customer records that have a `Name` that begins with an s. The FIND also places an EXCLUSIVE-LOCK on each record as it is read. This lock is released at the end of the REPEAT block.

In the output of this procedure, all the Customer names begin with s and the customers are displayed in order by country and then postal code.

Notes

- Specifying a field list (*`field-list`*) for *`record`* can increase the performance of remote (network) record retrieval substantially over specifying *`record`* alone.
- You cannot specify field lists or joins in a FIND statement, or specify field lists in an OPEN QUERY statement.
- You cannot reference a BLOB or CLOB field in a WHERE clause.
- Temp-tables and work tables can be used in join conditions specified with the OF option as long as the OF option requirements identified earlier in this section have been satisfied.
- Do not compare case-sensitive data with case-insensitive data in a WHERE expression. The AVM both cannot determine the results and does not raise the ERROR condition if you specify data with mixed case sensitivity in selection criteria because:
  - Mixed case sensitivity in selection criteria is handled differently by different DataServers.
  - Mixed case-sensitivity results for the same DataServer can be different depending on whether the query is resolved on the client or the server.
  - Some national languages do not support the concept of case sensitivity.

  Thus, such queries cannot be reliably resolved in any way.
- For SpeedScript, the only invalid option is USING FRAME.

- In a class definition, ABL treats the default buffer of a database table that you reference as an instance data member of the class. Therefore, you cannot access a default database buffer from a static member of a class, such as a static query or method; instead, you can define an alternate static buffer data member to access records of the database table from another static class member.
- You can reference any of the following functions and attributes in the *expression* of a WHERE or TENANT-WHERE option as long as the buffer specified for the function or attribute is **not** the same as the buffer of the query or FOR EACH statement:
  - BUFFER-GROUP-ID attribute
  - BUFFER-GROUP-ID function
  - BUFFER-GROUP-NAME attribute
  - BUFFER-GROUP-NAME function
  - BUFFER-TENANT-ID attribute
  - BUFFER-TENANT-ID function
  - BUFFER-TENANT-NAME attribute
  - BUFFER-TENANT-NAME function

See also

DEFINE QUERY statement, DO statement, FIND statement, FOR statement, OPEN QUERY statement, REPEAT statement

---

RECORD-LENGTH function

Returns the length of a record in a buffer as an INTEGER value.

Syntax

```
RECORD-LENGTH ( buffer )
```

buffer

A database buffer containing a record.

Notes

The RECORD-LENGTH function is especially useful when implementing ABL-based database replication, which involves storing entire database records in log record fields. ABL limits records to 32K. Before you transfer a record to a raw field in another record, you can use RECORD-LENGTH to ensure that you are not expanding the record beyond the 32K limit.

See also

RAW-TRANSFER statement

REJECTED function

Returns the current REJECTED attribute setting for a ProDataSet temp-table buffer.

Syntax

```
REJECTED( buffer-name )
```

buffer-name

The name of a ProDataSet temp-table buffer.

Notes

- This function is typically used with the SAVE-ROW-CHANGES( ) method.
- The REJECTED function corresponds to the REJECTED attribute.
- You can invoke the REJECTED function from within a WHERE clause (unlike the corresponding attribute).

RELEASE statement

Verifies that a record complies with mandatory field and unique index definitions. It clears the record from the buffer and unites it to the database if it has been changed.

Syntax

```
RELEASE record [ NO-ERROR ]
```

record

The name of a record buffer.

To use RELEASE with a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

For the RELEASE statement, as an exception to the general rule, if a unique index constraint is violated the NO-ERROR clause is ignored when validating the record. An error is raised before any changes are saved to the record to avoid damaging any indexes.

Example

The following example uses a browse widget to scan Customer records. Records within the browse are read with NO-LOCK. If you choose the Update Customer button, the CHOOSE trigger starts a transaction and applies an EXCLUSIVE-LOCK to the Customer record. When you have completed any updates, the procedure displays the new values in the browse widget and then executes a RELEASE statement. This ensures that the lock is released when the transaction ends.

r-rels.p

```
DEFINE VARIABLE methRtn   AS LOGICAL NO-UNDO.
DEFINE VARIABLE curr-cust AS ROWID   NO-UNDO.

DEFINE BUTTON upd-cust LABEL "Update Customer".
DEFINE BUTTON exit-app LABEL "Exit".

DEFINE QUERY seq-cust FOR customer.
DEFINE BROWSE brow-cust QUERY seq-cust
  DISPLAY Customer.CustNum Customer.Name WITH 10 DOWN.

FORM
  upd-cust exit-app SKIP(1)
  brow-cust
  WITH FRAME main-frame.
FORM
  Customer EXCEPT Customer.Comments
  WITH FRAME curr-frame COLUMN 40.

OPEN QUERY seq-cust FOR EACH Customer.

ON VALUE-CHANGED OF brow-cust DO:
  DISPLAY Customer EXCEPT Customer.Comments
    WITH FRAME curr-frame SIDE-LABELS.
  curr-cust = ROWID(Customer).
END.

ON CHOOSE OF upd-cust DO: /* TRANSACTION */
  FIND Customer WHERE ROWID(Customer) = curr-cust EXCLUSIVE-LOCK.
  UPDATE Customer WITH FRAME cust-frame VIEW-AS DIALOG-BOX
  TITLE "Customer Update".
  methRtn = brow-cust:REFRESH().
  DISPLAY Customer EXCEPT Customer.Comments
    WITH FRAME curr-frame SIDE-LABELS.
  RELEASE Customer.
END.

ENABLE ALL WITH FRAME main-frame.
APPLY "VALUE-CHANGED" TO brow-cust.
PAUSE 0 BEFORE-HIDE.
WAIT-FOR CHOOSE OF exit-app OR WINDOW-CLOSE OF DEFAULT-WINDOW.
```

If you omit the RELEASE statement in this example, the EXCLUSIVE-LOCK is downgraded to a SHARE-LOCK at the end of the transaction. This prevents other uses from updating that record. The SHARE-LOCK is released when you change the iteration of the browse.

Notes

- An ERROR occurs if the validation of the record fails. This can happen only with newly created records.
- If a record has been modified, the RELEASE statement causes a WRITE event and fires any related WRITE trigger to execute. All WRITE triggers execute before the record is actually written. If a WRITE trigger fails (or executes a RETURN statement with the ERROR option), the corresponding record is not written or released and the ERROR condition is raised for the RELEASE statement.

- See *ABL Essentials* for more information on transactions.

See also

NO-ERROR option

RELEASE EXTERNAL statement

Frees (that is, unloads from memory) a dynamic link library (DLL) or UNIX shared library.

Syntax

```
RELEASE EXTERNAL [ PROCEDURE ] "dll-name"
```

[PROCEDURE]

An optional "noise" keyword that does not affect the statement's behavior in any way.

dll-name

A character string representing the name of the DLL or UNIX shared library.

Example

To free the DLL, `mystuff.dll`, code the following statement:

```
RELEASE EXTERNAL PROCEDURE "mystuff.dll".
```

RELEASE OBJECT statement

Releases the specified COM object (Automation object or ActiveX control) and removes all internal structures associated with the handle to the object.

Syntax

```
RELEASE OBJECT COM-hdl-var [ NO-ERROR ]
```

COM-hdl-var

A COM-HANDLE variable that references a valid COM object.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

This procedure fragment shows a control named hc_CmdButton being loaded into a control-frame and the handle to the control (controlHdl) being obtained using the control name (hc_CmdButton) property. Later, it releases the control and deletes the parent control-frame widget (CFWidHdl).

```
DEFINE VARIABLE CFWidHdl   AS HANDLE     NO-UNDO.
DEFINE VARIABLE CFComHdl   AS COM-HANDLE NO-UNDO.
DEFINE VARIABLE controlHdl AS COM-HANDLE NO-UNDO.

/* Create frame foo ... */
CREATE CONTROL-FRAME CFWidHdl ASSIGN
  FRAME = FRAME foo:HANDLE
  NAME  = "ctlFrame1".

CFComHdl = CFWidHdl:COM-HANDLE.
CFComHdl:LoadControls(hc_CmdButton.wrx, "hc_CmdButton").

controlHdl = CFComHdl:hc_CmdButton.
controlHdl:BgColor = RGB-VALUE(0,128,0).

/* Do some more stuff ... WAIT-FOR ... */
RELEASE OBJECT controlHdl. /* NOTE: Not really necessary */
DELETE WIDGET CFWidHdl.
```

For an example of the RELEASE OBJECT statement applied to Automation objects, see the CREATE automation object statement entry.

Notes

- After this statement completes, any other component handles that reference the object are invalid. If you attempt to reference the object using one of these handles, the AVM returns an invalid handle error. It is also possible for a newly instantiated COM object to get the same handle as one that has been released. The AVM does not detect that this occurs. In this case, the "old" handle is valid, but it references a different control. Thus, it is a good practice to set any COM-HANDLE variables that reference a released COM object to the Unknown value (?).
- The released COM object remains active as long as any other COM object has a valid reference to it. In the case of an ActiveX control, the parent control-frame is a COM object that references the control. All other component handle references you establish in the ABL session represent a second reference to the COM object. Thus, when you release one of these component handles, the released COM object remains active as long as the parent control-frame COM object is still active. To release the parent control-frame COM object and complete the release of the ActiveX control, you must follow any release of the ActiveX control by a delete of the parent control-frame widget.
- When you delete a control-frame widget, the AVM releases all associated ActiveX controls automatically, whether or not you release them individually.
- When the session ends, the AVM automatically releases any active COM objects you have not released individually.

See also

CREATE automation object statement, DELETE WIDGET statement, DELETE WIDGET-POOL statement, NO-ERROR option

REPEAT statement

Begins a block of statements that are processed repeatedly until the block ends. An END statement is used to terminate a REPEAT block.

Syntax

This is the syntax for the REPEAT block:

```
[ label : ]
REPEAT
  [ FOR record[ , record ]...]
  [ preselect-phrase ]
  [ query-tuning-phrase ]
  [ variable = expression1 TO expression2 [ BY k ]]
  [ WHILE expression ]
  [ TRANSACTION ]
  [ stop-after-phrase ]
  [ on-endkey-phrase ]
  [ on-error-phrase ]
  [ on-quit-phrase ]
  [ on-stop-phrase ]
  [ frame-phrase ] :
  <block of statements>
END.
```

FOR *record*[, *record*] . . .

Names a record buffer and scopes the buffer to the block. The scope of a record determines when the buffer is cleared and the record written back to the database. See *ABL Essentials* for more information on record scoping and blocks.

To access a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See Record phrase for more information.

preselect-phrase

Goes through a table to select the records that meet the criteria you specify in a *record-phrase*. PRESELECT creates a temporary index that contains pointers to each of the preselected records in the database table. You can then use other statements, such as FIND NEXT, to process those records. This is the syntax for the *preselect-phrase*:

```
PRESELECT
  [ EACH | FIRST | LAST ]record-phrase
    [ , { EACH | FIRST | LAST }record-phrase]...
  [[ BREAK ]{ BY expression[ DESCENDING ]}...]
```

For more information, see PRESELECT phrase.

query-tuning-phrase

Allows programmatic control over the execution of a DataServer query. This is the syntax for the *query-tuning-phrase*:

```
QUERY-TUNING
  (
    [ BIND-WHERE | NO-BIND-WHERE ]
    [ CACHE-SIZE integer]
    [ DEBUG { SQL | EXTENDED }| NO-DEBUG ]
    [ INDEX-HINT | NO-INDEX-HINT ]
    [ JOIN-BY-SQLDB | NO-JOIN-BY-SQLDB ]
    [ LOOKAHEAD | NO-LOOKAHEAD ]
    [ SEPARATE-CONNECTION | NO-SEPARATE-CONNECTION ]
  )
```

For more information see the OpenEdge DataServer Guides (*Use the Microsoft SQL Data Server* and *Use the Microsoft SQL Data Server*).

variable = *expression1* TO *expression2* [BY *k*]

Indicates the name of a field or variable whose value you are incrementing in a loop. *expression1* is the starting value for *variable* on the first iteration of the loop. *k* is the amount to add to *variable* after each iteration and must be a constant. When *variable* exceeds *expression2* (or is less than *expression2* if *k* is negative), the loop ends. Because *expression1* is compared to *expression2* at the start of the first iteration of the block, the block can be executed zero times. *expression2* is reevaluated with each iteration of the block.

WHILE *expression*

The REPEAT block iterates as long as the condition specified by *expression* is TRUE. *expression* is any combination of constants, field names, and variable names that yield a logical value.

TRANSACTION

Identifies the REPEAT block as a system transaction block. The AVM starts a system transaction for each iteration of a transaction block if there is no active system transaction. See *ABL Essentials* for more information on transactions.

stop-after-phrase

Specifies a time-out value (in seconds) for the block. This is the syntax for the STOP-AFTER phrase:

```
STOP-AFTER time-limit
```

For more information see STOP-AFTER phrase.

on-endkey-phrase

Describes the processing that takes place when the **ENDKEY** condition occurs during a block. This is the syntax for the ON ENDKEY phrase:

```
ON ENDKEY UNDO
  [ label1 ]
  [      , LEAVE [ label2 ]
       | , NEXT [ label2 ]
       | , RETRY [ label1 ]
       | , RETURN [ return-value |
                    ERROR [ return-value | error-object-expression ] |
                    NO-APPLY ]
```

For more information see the ON ENDKEY phrase.

on-error-phrase

Describes the processing that takes place when there is an error during a block. This is the syntax for the ON ERROR phrase:

```
ON ERROR UNDO
  [ label1 ]
  [      , LEAVE [ label2 ]
       | , NEXT [ label2 ]
       | , RETRY [ label1 ]
       | , RETURN [ return-value |
                    ERROR [ return-value | error-object-expression ] |
                    NO-APPLY ]
       | , THROW
  ]
```

For more information see the ON ERROR phrase.

on-quit-phrase

Describes the processing that takes place when a QUIT statement is executed during a block. This is the syntax for the ON QUIT phrase:

```
ON QUIT
  [ UNDO [ label1 ] ]
  [      , LEAVE [ label2 ]
       | , NEXT [ label2 ]
       | , RETRY [ label1 ]
       | , RETURN [ return-value |
                    ERROR [ return-value | error-object-expression ] |
                    NO-APPLY ]
  ]
```

For more information see the ON QUIT phrase.

on-stop-phrase

Describes the processing that takes place when the STOP conditions occurs during a block. This is the syntax for the ON STOP phrase:

```
ON STOP UNDO
  [ label1 ]
  [      , LEAVE [ label2 ]
       | , NEXT [ label2 ]
       | , RETRY [ label1 ]
       | , RETURN [ return-value |
                    ERROR [ return-value | error-object-expression ] |
                    NO-APPLY ]
  ]
```

For more information see the ON STOP phrase.

frame-phrase

Specifies the overall layout and processing properties of a frame. This is the syntax for the frame phrase:

```
WITH [ ACCUM [ max-length ] ]
     [ at-phrase ]
     [ CANCEL-BUTTON button-name ] [ CENTERED ]
     [ color-specification ]
     [ COLUMN expression ] [ n COLUMNS ]
     [ CONTEXT-HELP ] [ CONTEXT-HELP-FILE help-file-name ]
     [ DEFAULT-BUTTON button-name ]
     [ DROP-TARGET ]
     [ [expression] DOWN ] [ EXPORT ]
     [ WIDGET-ID id-number ] [ FONT expression ]
     [ FRAME frame ]
     [ INHERIT-BGCOLOR | NO-INHERIT-BGCOLOR ]
     [ INHERIT-FGCOLOR | NO-INHERIT-FGCOLOR ]
     [ KEEP-TAB-ORDER ] [ NO-BOX ]
     [ NO-HIDE ] [ NO-LABELS ] [ USE-DICT-EXPS ]
     [ NO-VALIDATE ] [ NO-AUTO-VALIDATE ]
     [ NO-HELP ] [ NO-UNDERLINE ]
     [ OVERLAY ] [ PAGE-BOTTOM | PAGE-TOP ] [ RETAIN n ]
     [ ROW expression ] [ SCREEN-IO | STREAM-IO ]
     [ SCROLL n ] [ SCROLLABLE ] [ SIDE-LABELS ]
     [ size-phrase ] [ STREAM stream | STREAM-HANDLE handle ] [ THREE-D ]
     [ title-phrase ] [ TOP-ONLY ] [ USE-TEXT ]
     [ V6FRAME [ USE-REVVIDEO | USE-UNDERLINE ] ]
     [ VIEW-AS DIALOG-BOX ] [ WIDTH n ] [ IN WINDOW window ]
```

For more information, see the Frame phrase.

Example

In this menu procedure, if you press **END-ERROR** or **ENDKEY** when prompted for your menu selection, any selection data you have entered is undone and the procedure continues to prompt you:

r-rpt.p

```
DEFINE VARIABLE Selection AS INTEGER NO-UNDO FORMAT "9".

FORM SKIP(3)
  "0 - Exit" at 32
  "1 - Edit Customer File" at 32
  "2 - List Customer File" at 32
  "3 - Edit Item File" at 32
  "4 - List Item File" at 32
  "Enter Choice" TO 30 Selection AUTO-RETURN
  HEADER "Application Name"  "Master Menu" AT 34  "Company" TO 79
  WITH NO-BOX NO-LABELS CENTERED FRAME menu.

/* Create the procedures that are called from the following block. */
REPEAT ON ENDKEY UNDO, RETRY:
  UPDATE Selection WITH FRAME menu.
  HIDE FRAME menu.

  CASE(Selection):
    WHEN 0 THEN LEAVE.
    WHEN 1 THEN RUN custedit.p.
    WHEN 2 THEN RUN custrpt.p.
    WHEN 3 THEN RUN itemedit.p.
    WHEN 4 THEN RUN itemrpt.p.
    OTHERWISE DO:
      BELL.
      MESSAGE "Not a valid choice. Try again.".
    END.
  END CASE.
END. /* REPEAT */
```

Notes

- Within a REPEAT block, if you are using the FIND NEXT or FIND PREV statement and you change the value of an index field, the AVM makes that change in the index table at the end of the UPDATE or SET statement. Therefore, if you change the value so that the record appears later in the index table, you see the record again if you FIND NEXT. If you change the value so that the record appears earlier in the index table, you see the record again if you FIND PREV.

  ```
  REPEAT:
    FIND NEXT Customer.
    UPDATE Customer.CustNum.
  END.
  ```

  In this example, if you change Customer 1 to Customer 300, you see that Customer record again at the end of the procedure.

  When you use the PRESELECT option, the AVM builds a special index table that is **not** updated when index values change:

  ```
  REPEAT PRESELECT EACH Customer:
    FIND NEXT Customer.
    UPDATE Customer.CustNum.
  END.
  ```

  In this example, if you change Customer 2 to Customer 200, you do not see that Customer record until you look it up with a new procedure.

- For SpeedScript, the *on-endkey-phrase* and *on-quit-phrase* are not valid options.

See also

DO statement, END statement, Frame phrase, ON ENDKEY phrase, ON ERROR phrase, ON QUIT phrase, ON STOP phrase, Record phrase, STOP-AFTER phrase

REPLACE function

Returns a string with specified substring replacements. The data type of the returned value matches the data type of the expression passed to the function.

Syntax

```
REPLACE ( source-string , from-string , to-string )
```

source-string

Specifies the base string to make replacements in. The *source-string* parameter can be any expression that evaluates to a string or a LONGCHAR. The REPLACE function does not change the value of *source-string* itself; the function returns the string with replacements.

from-string

Specifies the substring to replace. The *from-string* parameter can be any expression that evaluates to a string or a LONGCHAR. Each occurrence of *from-string* within *source-string* is replaced.

to-string

Specifies the replacement substring. The *to-string* parameter can be any expression that evaluates to a string or a LONGCHAR. Each occurrence of *from-string* in *source-string* is replaced by *to-string*.

Example

The following example uses the REPLACE function to replace the string "user" with an actual user ID, if available:

r-repl.p

```
DEFINE VARIABLE greeting AS CHARACTER NO-UNDO FORMAT "x(40)"
  INITIAL "Starting user's session . . . ".

IF USERID("DICTDB") < > "" THEN
  greeting = REPLACE(greeting, "user", USERID("DICTDB")).
DISPLAY greeting WITH NO-LABELS.
```

Notes

- The REPLACE function replaces all occurrences of *from-string* within *source-string*. After replacing a substring, the REPLACE function resumes searching the string after the inserted text. Thus, the inserted text is not recursively searched (in whole or in part) for *from-string*.
- The search for occurrences of *from-string* within *source-string* is not case sensitive, unless one of the three values used in the function (*source-string*, *to-string*, or *from-string*) is a case-sensitive field or variable.

See also

OVERLAY statement, SUBSTITUTE function, SUBSTRING function

REPOSITION statement

Repositions the cursor associated with a specific query. The query must be associated with a browse widget or defined with the SCROLLING option. The next record to be retrieved is the record following the cursor position.

Syntax

```
REPOSITION query
  {     TO ROWID rowid1 [ , rowid2 ] . . .
          [ FOR TENANT tenant-expression ] [ NO-ERROR ]
    |   TO RECID recid [ NO-ERROR ]
    |   TO ROW n
    |   FORWARDS n
    |   BACKWARDS n
  }
```

query

The name of the query to reposition. The query must be open.

TO ROWID *rowid1* [, *rowid2*] . . . [FOR TENANT *tenant-expression*] [NO-ERROR]

Repositions the join levels of a query to the corresponding ROWID expressions (*rowid1*, *rowid2*, and so on) that you specify, where *rowid1* represents the ROWID for the top level of the join, *rowid2* represents the ROWID for the next level of the join, and so on. You can specify any number of ROWID expressions up to the number of join levels. If you specify fewer ROWID expressions than the number of join levels, the AVM repositions the join levels of the query to the corresponding ROWID expressions you specify, but positions the remaining join levels for the unspecified ROWID expressions arbitrarily.

The FOR TENANT option is useful only for a multi-tenant database, and primarily one with a connection identity that has super tenant access. If the user has a super-tenant connection identity and you **do not** specify this option, the query repositions to data owned by the effective tenant. If you **do** specify this option, the query repositions to data owned by the regular tenant identified by *tenant-expression*.

If the user has a regular-tenant connection identity, and you specify this option, *tenant-expression* must match the tenancy of the connection identity. Otherwise, the statement raises ERROR.

If *tenant-expression* evaluates to an integer, the value must be a valid tenant ID for a regular tenant or zero (0) for the default tenant. If *tenant-expression* evaluates to a character string, the value must be a valid tenant name for a regular or "Default" for the default tenant. Otherwise, the statement raises ERROR.

TO RECID *recid* [NO-ERROR]

Similar to the TO ROWID option, except that the value *recid* is an expression that evaluates to a RECID value, and you can specify only one recid. Supported only for backward compatibility.

NO-ERROR suppresses any error messages that result from specifying an illegal value or a value that does not identify any records returned by the query. See the NO-ERROR entry below for more information.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

TO ROW *n*

Repositions the cursor to before the specified row in the result list of the query. The value *n* must be an integer expression that identifies a row in the result list. You cannot use this option with a query opened with the INDEXED-REPOSITION option.

FORWARDS *n*

Moves the cursor from its current position in the result list to a new position *n* records forward, where *n* represents an integer expression.

REPOSITION FORWARDS always places the cursor between two rows. For example:

- If the cursor is on a row—say, row 5—REPOSITION FORWARDS 1 moves the cursor to row 6, then to half way between rows 6 and 7. From this position, GET PREVIOUS moves the cursor to row 6, while GET-NEXT moves the cursor to row 7.
- If the cursor is already between two rows—say, between rows 5 and 6— REPOSITION FORWARDS 1 moves the cursor to half way between rows 6 and 7. From this position, GET PREVIOUS moves the cursor to row 6, while GET-NEXT moves the cursor to row 7.

BACKWARDS *n*

Moves the cursor from its current position in the result list to a new position *n* records back, where *n* represents an integer expression.

REPOSITION BACKWARDS always places the cursor between two rows. For example:

- If the cursor is on a row—say, row 5—REPOSITION BACKWARDS 1 moves the cursor to row 4, then to half way between rows 4 and 5. From this position, GET PREVIOUS moves the cursor to row 4, while GET-NEXT moves the cursor to row 5.
- If the cursor is already between two rows—say, between rows 5 and 6— REPOSITION BACKWARDS 1 moves the cursor to half way between rows 4 and 5. From this position, GET PREVIOUS moves the cursor to row 4, while GET-NEXT moves the cursor to row 5.

Example

The following example uses the REPOSITION statement to move forward or backward within a query:

r-repos.p

```
DEFINE VARIABLE num AS INTEGER NO-UNDO INITIAL 1.

DEFINE QUERY q-order FOR Customer, Order SCROLLING.

DEFINE BUTTON b_quit LABEL "Quit".
DEFINE BUTTON b_frwd LABEL "FORWARD".
DEFINE BUTTON b_back LABEL "BACKWARD".

FORM b_frwd b_back b_quit
  WITH FRAME butt-frame ROW 1.

ON CHOOSE OF b_back, b_frwd DO:
  PROMPT-FOR num LABEL "Records To Skip"
    WITH FRAME pos-info CENTERED ROW 5 overlay.
  HIDE FRAME pos-info NO-PAUSE.
  IF SELF:LABEL = "BACKWARD" THEN
    REPOSITION q-order BACKWARDS INPUT num + 1.
  ELSE
    REPOSITION q-order FORWARDS INPUT num - 1.
  RUN getone.
END.

OPEN QUERY q-order FOR EACH Customer NO-LOCK,
  EACH Order OF Customer NO-LOCK.
RUN getone.

ENABLE b_back b_frwd b_quit WITH FRAME butt-frame.
WAIT-FOR CHOOSE OF b_quit OR WINDOW-CLOSE OF CURRENT-WINDOW.

PROCEDURE getone:
  GET NEXT q-order.
  IF NOT AVAILABLE Customer THEN DO:
    REPOSITION q-order BACKWARDS 1.
    GET NEXT q-order.
  END.
  DISPLAY Customer.CustNum Customer.Name SKIP
    Order.OrderNum Order.OrderDate
    WITH FRAME order-info CENTERED ROW 5 SIDE-LABELS OVERLAY.
END PROCEDURE.
```

Notes

- The REPOSITION statement does not fetch a record, except when the query is associated with a browse. The REPOSITION statement positions the cursor for the query so that a subsequent GET NEXT statement fetches the specified record, and GET PREV fetches the record before it.
- After executing a REPOSITON statement that involves a multi-table join, the bottom-most buffer will not be available, as is the case for a query built on a single table. You then need to execute a GET NEXT statement to make the row you want available. The availability of non-bottom level buffers following the REPOSITION, however, is undetermined. That is, non-bottom level buffers may or may not be available.
- If you reposition a query associated with a browse widget, the browse widget data is refreshed with the record after the new position at the top.
- If you try to position the cursor outside the list of records that satisfy the query, the AVM does **not** raise the ERROR condition. If you try to position the cursor before the first record, the AVM positions the query to just before the first record. If you try to position the cursor beyond the last record, the AVM positions it just beyond the last record.
- The REPOSITION statement might be slow if the record you position to has not yet been fetched.

- The REPOSITION TO ROWID statement might be especially slow. If the record has not yet been fetched, the AVM performs a series of GET NEXT operations until the record is found. You can optimize the performance of a REPOSITION TO ROWID statement by opening the query using the INDEXED-REPOSITION option of the OPEN QUERY statement.
- The INDEXED-REPOSITION option of the OPEN QUERY statement, followed by REPOSITION TO ROWID or GET LAST, causes the query results list to change dramatically. Subsequent use of the CURRENT-RESULT-ROW or NUM-RESULTS functions might produce unknown or unexpected results.
- The order of the records in the query is determined by the options specified in the OPEN QUERY statement.
- For SpeedScript, the *on-endkey-phrase* and the *on-quit-phrase* do not apply.
- When specifying the FOR TENANT option, the AVM looks up *tenant-expression* in the database with a share lock. The AVM waits 60 seconds to get the share lock and raises ERROR if it fails to obtain the share lock in that amount of time. The AVM releases the share lock immediately after successfully fetching the row. This share lock is released even if the statement is called while in the scope of a transaction.

See also

CLOSE QUERY statement, CURRENT-RESULT-ROW function, DEFINE QUERY statement, GET statement, NO-ERROR option, NUM-RESULTS function, OPEN QUERY statement

RETRY function

Returns a TRUE value if the current block is being reprocessed after a previous UNDO, RETRY.

Syntax

```
RETRY
```

Example

```
DEFINE VAR badState AS CHAR NO-UNDO.

FOR EACH Customer:
    IF RETRY THEN
    DO:
        /* So user can see the bad value they typed */
        Customer.State = badState.
    END.
    UPDATE City State PostalCode.
    IF NOT CAN-FIND(State WHERE State.State = Customer.State) THEN DO:
        MESSAGE "This is not a valid state"
            VIEW-AS ALERT-BOX INFORMATION BUTTONS OK.
        badState = Customer.State.
        UNDO, RETRY.
    END.
END.
```

Notes

- Using the RETRY function in a block turns off the default error processing, which result in no infinite loop protection for the block.

- For more information on retry processing, see *ABL Essentials*.

See also

UNDO statement

RETURN statement

Leaves the local or remote procedure, user-defined function, ON trigger, method of a class, class constructor or destructor, or property accessor, and returns to the caller. If there is no caller, returns to the Procedure Editor or Progress Developer Studio for OpenEdge tool.

Syntax

```
RETURN
    [ return-value |
      ERROR [ error-return-value | error-object-expression ] |
      NO-APPLY ]
```

return-value

This has different behavior depending on whether a value can be returned from the current context. From contexts that don't return a value, this is used to set RETURN-VALUE.

For a procedure or ON trigger block, the value is used to set RETURN-VALUE:

- If *return-value* is specified, it must be a CHARACTER expression.
- If *return-value* is not specified, RETURN-VALUE is set to the empty string ("").
- To retrieve the *return-value*, access the RETURN-VALUE function from the caller.

For a user-defined function, non-VOID method, or property accessor:

- *Return-value* must be specified.
- *Return-value* must be an expression whose data type matches the return type of the function, method, or property; data type matching between the expression and return type follows the data type widening rules for an expression passed to an OUTPUT parameter (see the Parameter passing syntax entry for more information).
- To retrieve the *return-value* from the caller, simply reference the function or method call in an expression.

For a VOID method, you cannot set a *return-value*.

ERROR

Causes an ERROR condition in the calling block.

You **cannot** use the ERROR option in a user-interface trigger block to raise ERROR outside of the trigger block.

RETURN ERROR in a user-defined function does not raise ERROR in the caller. Instead, it sets the target variable of the function to the Unknown value (?). See *ABL Error Handling* for more information.

error-return-value

Any CHARACTER expression.

To retrieve the return value in the caller, catch the Progress.Lang.AppError using a CATCH statement and obtain the ReturnValue property of the AppError. Alternatively, the caller can access the RETURN-VALUE function.

If you do not specify either *error-return-value* or *error-object-expression*, both the RETURN-VALUE function as well as the AppError ReturnValue property is set to the empty string("").

error-object-expression

An expression that resolves to a specific error object. It must be an object derived from Progress.Lang.ProError or an object of a class that implements Progress.Lang.Error. The compiler generates an error if these rules are not met. Note that the only error object you can instantiate directly is a Progress.Lang.AppError object or a subclass.

RETURN ERROR *error-object-expression* immediately returns to the caller before throwing the error object.

NO-APPLY

Suppresses the default behavior for the current user-interface event. You thus can use the NO-APPLY option in a user-interface trigger block to suppress that behavior. For example, the default behavior for a character code key press in a fill-in field is to echo the character in the field. If you execute RETURN NO-APPLY in a trigger, this behavior is not performed. Also, NO-APPLY returns without setting a *return-value* or error object.

Examples

The following code demonstrates using RETURN in a user-defined function.

```
DEFINE VARIABLE dTemp AS DECIMAL NO-UNDO INITIAL 0.

FUNCTION CtoF RETURNS DECIMAL (INPUT dCelsius AS DECIMAL):
    RETURN (dCelsius * 1.8) + 32.
END.

MESSAGE dTemp "degrees Celsius is" CtoF(dTemp) "degrees Fahrenheit." SKIP
    VIEW-AS ALERT-BOX.
```

Output:

```
0 degrees Celsius is 32 degrees Fahrenheit.
```

The following code illustrates using RETURN ERROR in a user-defined function.

```
DEFINE VARIABLE iFuncReturn AS INTEGER INITIAL 99 NO-UNDO.

FUNCTION ErrorTest RETURNS INTEGER:
    RETURN ERROR.
END FUNCTION.

ASSIGN iFuncReturn = ErrorTest().
IF iFuncReturn EQ ? THEN
    MESSAGE "Error in user-defined function." view-as alert-box.
```

Output:

```
Error in user-defined function.
```

The following code demonstrates using RETURN ERROR with an error object.

```
DEF VAR ix AS INT.

RUN proc NO-ERROR.
IF ERROR-STATUS:ERROR THEN
DO:
    DO ix = 1 TO ERROR-STATUS:NUM-MESSAGES: /* this will be 2 */
        MESSAGE ERROR-STATUS:GET-MESSAGE(ix) ERROR-STATUS:GET-NUMBER(ix).
    END.
END.

PROCEDURE proc:
    DEFINE VAR err AS PROGRESS.Lang.AppError.

    err = NEW PROGRESS.Lang.AppError("The car cannot be rented",1).
    err:AddMessage ("No driver's license was provided", 98).

    RETURN ERROR err.
END.
```

Output:

```
The car cannot be rented 1
No driver's license was provided 98
```

Notes

- The RETURN-VALUE function provides the value set by the most recently executed RETURN statement.
- If a procedure executing the RETURN statement is called asynchronously, the client can access the *return-value* and ERROR condition in the associated event procedure or method.
- Class-based error objects can be thrown from an application server and handled by a CATCH block on an ABL client. To be throwable from an application server to an ABL client, user-defined error classes must be defined on both the server and client sides, and the classes must be defined as SERIALIZABLE. For the full list of restrictions on class-based objects that are passed between application server and client, see the Parameter passing syntax entry. For more information on error handling in general, see *ABL Error Handling*.

- Class-based error objects can also be thrown from an application server to a client for an asynchronous request. Using `RETURN ERROR` is one way to do this. In this case, the error object is returned to the client via the `ERROR-OBJECT` attribute of the Asynchronous request object handle.

See also

CONSTRUCTOR statement, DEFINE PROPERTY statement, ERROR-OBJECT attribute, FUNCTION statement, METHOD statement, ON ENDKEY phrase, ON ERROR phrase, ON QUIT phrase, ON STOP phrase, RETURN-VALUE function, UNDO statement

RETURN-VALUE function

Provides a character string value returned by the most recently executed RETURN statement of a local or remote procedure, trigger block, user-defined function, method of a class, class constructor, or property accessor.

Syntax

```
RETURN-VALUE
```

Example

For examples of the RETURN-VALUE function, see the RETURN statement reference entry.

Notes

- The returned value has the CHARACTER data type.
- When you access RETURN-VALUE, its value represents the value returned by the most recently executed RETURN or THROW:
  - If it is a RETURN with options that can set the RETURN-VALUE, but does not, the empty string ("") is returned.
  - If it is a THROW that returns a Progress.Lang.AppError object, the value of the object's ReturnValue property is returned.
- To reliably access a RETURN-VALUE setting returned from a block or called routine, check RETURN-VALUE as soon as you can after the block or called routine terminates.
- RETURN-VALUE does not return the value of a non-VOID method of a class or user-defined function invocation. You return the values of these methods and functions within an expression where you reference their invocations, similar to referencing a variable value.

See also

CREATE SERVER statement, ON ENDKEY phrase, ON ERROR phrase, RETURN statement, UNDO statement

RGB-VALUE function

Returns an INTEGER value that represents a combination of a red, green, and blue color value. This function allows you to define an arbitrary color, expanding beyond those colors defined in the color table.

Note: Does not apply to SpeedScript programming.

Syntax

```
RGB-VALUE ( redval , greenval , blueval )
```

redval, greenval, blueval

Identifies red, green, and blue color values which can be combined to define a unique color value.

Example

The following code fragment shows how to set the background color of an ActiveX control:

```
DEFINE VARIABLE hdlControl AS COM-HANDLE NO-UNDO.

/* Complete code to get a handle to a control in a control-frame.*/
. . .
hdlControl:BackColor = RGB(128, 0, 256).
```

For detailed information on programming ActiveX Controls, see *OpenEdge Programming Interfaces*.

Notes

The RGB-VALUE function is generally most useful when it is used with ActiveX Controls.

See also

COLOR-TABLE system handle

RIGHT-TRIM function

Removes trailing white space, or other specified characters, from a CHARACTER or LONGCHAR expression. The data type of the returned value matches the data type of the expression passed to the function.

Syntax

```
RIGHT-TRIM ( expression [ , trim-chars ] )
```

expression

An expression (a constant, field name, variable name, or expression) whose value is a CHARACTER or LONGCHAR. If *expression* is a case-sensitive variable, the AVM performs a case-sensitive trim. If *expression* is a LONGCHAR, the result is in the same code page.

trim-chars

A character expression that specifies the characters to trim from *expression*. If you do not specify *trim-chars*, the RIGHT-TRIM function removes spaces, tabs, line feeds, and carriage returns.

Example

The following example shows the effects of the TRIM, RIGHT-TRIM, and LEFT-TRIM functions:

r-ltrim.p

```
DEFINE VARIABLE ix  AS INTEGER   NO-UNDO.
DEFINE VARIABLE txt AS CHARACTER NO-UNDO FORMAT "X(26)"
  INITIAL "***** This is a test *****".

DEFINE BUTTON b_left  LABEL "Left Trim".
DEFINE BUTTON b_right LABEL "Right Trim".
DEFINE BUTTON b_trim  LABEL "Trim".
DEFINE BUTTON b_quit  LABEL "Quit" AUTO-ENDKEY.

DEFINE FRAME butt-frame
  txt ix LABEL "String Length" SKIP(2)
  b_left b_right b_trim b_quit
WITH CENTERED TITLE "Original Text String".

DEFINE FRAME trimed-frame
  txt LABEL "Trimed Text"
  ix  LABEL "Length"
  WITH CENTERED.

ON CHOOSE OF b_trim, b_right, b_left IN FRAME butt-frame DO:
  FRAME trimed-frame:TITLE  = "Data After " + SELF:LABEL.
  DISPLAY TRIM(txt, "* ") WHEN SELF:LABEL = "Trim" @  txt
    LENGTH(TRIM(txt, "* ")) WHEN SELF:LABEL = "Trim" @ ix
    LEFT-TRIM(txt,"* ") WHEN SELF:LABEL = "Left Trim"  @ txt
    LENGTH(LEFT-TRIM(txt,"* ")) WHEN SELF:LABEL = "Left Trim" @ ix
    RIGHT-TRIM(txt, "* ") WHEN SELF:LABEL = "Right Trim" @ txt
    LENGTH(RIGHT-TRIM(txt, "* ")) WHEN SELF:LABEL = "Right Trim"  @ ix
  WITH FRAME trimed-frame.
END.

ENABLE b_left b_right b_trim b_quit WITH FRAME butt-frame.

ix = LENGTH(txt).
DISPLAY txt ix WITH FRAME butt-frame.

WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame.
```

Notes

- The RIGHT-TRIM function is similar to the TRIM function except that it trims characters only from the right end of the string.
- If *expression* is a case-sensitive field or variable, then *trim-chars* is also treated as case sensitive. Otherwise, *trim-chars* is not case sensitive.
- The RIGHT-TRIM function is double-byte enabled. The specified *expression* and *trim-chars* arguments can contain double-byte characters. RIGHT-TRIM does not remove double-byte space characters by default.

See also

LEFT-TRIM function, TRIM function

ROUND function

Rounds a decimal expression to a specified number of places after the decimal point.

Syntax

```
ROUND ( expression , precision )
```

expression

A decimal expression.

precision

A non-negative integer expression whose value is the number of places you want in the decimal result of the ROUND function.

Example

This procedure increases all `CreditLimit` values by 10 percent, rounding those values to the nearest $100:

r-round.p

```
FOR EACH Customer:
  DISPLAY Customer.CustNum Customer.Name Customer.CreditLimit.
  Customer.CreditLimit = ROUND((Customer.CreditLimit * 1.1) / 100 ,0) * 100.
  PAUSE.
  DISPLAY Customer.CreditLimit.
END.
```

See also

TRUNCATE function

ROUTINE-LEVEL ON ERROR UNDO, THROW statement

Use this statement in a procedure (`.p`) or class (`.cls`) file to change the implicit, default `ON ERROR` directive from `UNDO, RETRY` to `UNDO, THROW` for all routine-level blocks in a file. This is specifically for `ERROR`, not `STOP` conditions because `STOP` conditions are already thrown by default. The statement must come before any executable or `DEFINE` statements in a file. However, it can come either before or after a `USING` statement. In a `.cls` file, the statement must come before the `CLASS` statement.

Syntax

```
ROUTINE-LEVEL ON ERROR UNDO, THROW.
```

The following blocks are the routine-level blocks affected by this statement:

- Main block of an external procedure (.p)
- Internal procedures
- User-defined functions
- Methods of a class
- Class constructors
- Property accessors
- ON blocks used as database triggers with CREATE, DELETE, WRITE or ASSIGN events

This statement does not affect:

- Destructors
- DO, FOR, or REPEAT blocks contained within the routine-level blocks
- ON blocks that are UI triggers.

Note these alternatives to the ROUTINE-LEVEL ON ERROR UNDO, THROW statement:

- Instead of adding the statement to source-code files, you can use the -undothrow 1 startup parameter to change the default error handling on routine-level blocks to UNDO, THROW during compilation. See the *Startup Command and Parameter Reference* for more information.
- The BLOCK-LEVEL ON ERROR UNDO, THROW statement can be used if you want to change the default error handling on REPEAT, FOR, and DO TRANSACTION blocks in addition to routine-level blocks. (You can use the -undothrow 2 startup parameter to change the default error-handling to UNDO, THROW on every block affected by the BLOCK-LEVEL statement during compilation.)

Example

This code propagates an error from an internal procedures up to the main procedure block.

r-ROUTINE-LEVEL-01.p

```
ROUTINE-LEVEL ON ERROR UNDO, THROW.

PROCEDURE find1000:
  FIND FIRST Customer WHERE Customer.CustNum = 1000.
END PROCEDURE.

PROCEDURE find2000:
  FIND FIRST Customer WHERE Customer.CustNum = 2000.
END PROCEDURE.

PROCEDURE find3000:
  FIND FIRST Customer WHERE Customer.CustNum = 3000.
END PROCEDURE.

/* Main Block */
RUN find1000.
RUN find2000.
RUN find3000.

CATCH eAnyError AS Progress.Lang.SysError:
  MESSAGE "Your CATCH block associated with the the main block (.p) has handled an error
 in an internal procedure."
    VIEW-AS ALERT-BOX BUTTONS OK.
END CATCH.
```

Notes

- The ROUTINE-LEVEL ON ERROR UNDO, THROW statement guarantees that all unhandled errors in a subprocedure of a persistent procedure, or method of a class, are propagated up to the caller. You decide for each subprocedure, or method within the file, whether that subprocedure or method should handle errors with its own CATCH block. Alternatively, you may want a CATCH block at the caller level to handle errors. This can be useful if a caller calls many internal procedures in a persistent procedure or many methods in a class.
- You cannot change the default behavior for individual routine-level blocks to throw or rethrow errors.
- Error objects can be thrown from an application server and handled by a CATCH block on an ABL client. To be throwable from an application server to an ABL client, user-defined error classes must be defined on both the server and client sides, and the classes must be defined as SERIALIZABLE. For the full list of restrictions on class-based objects that are passed between application server and client, see the Parameter passing syntax entry. For more information on error handling in general, see *ABL Error Handling*.

See also

BLOCK-LEVEL ON ERROR UNDO, THROW statement, ON ENDKEY phrase, ON ERROR phrase, ON QUIT phrase, RETURN statement, RETURN-VALUE function, STOP statement, UNDO-THROW-SCOPE attribute

ROW-STATE function

Returns an INTEGER value that represents the current change state of a static ProDataSet temp-table buffer.

Syntax

```
ROW-STATE( buffer-name )
```

buffer-name

The name of a ProDataSet temp-table buffer (preferably a before-image temp-table buffer).

Notes

- The ROW-STATE function corresponds to the ROW-STATE attribute.
- When the TRACKING-CHANGES attribute is set to TRUE for a ProDataSet temp-table, the AVM tracks changes to the data in that temp-table using a before-image temp-table that contains the original version of each modified row. You can think of the temp-table itself as the after-image because it contains the latest version of each row.

  Every row in the after-image table that has been modified or created corresponds to a row in the before-image table. Deleted rows do not appear in the after-image table, because it reflects the current state of the data. Every row in the before-image table has a non-zero ROW-STATE, because every row is the before-image of a deleted, created, or modified row in the after-image table. Unchanged rows do not appear in the before-image table.

  You can use the ROW-STATE function on each row in either the after-image table or the before-image table to determine whether a row has changed and how it has changed.

- The possible return values can be expressed as compiler constants. The following table lists these values.

Table 59: Row state values

Compiler constant	Value	Description
ROW-UNMODIFIED	0	The row was not modified.
ROW-DELETED	1	The row was deleted.
ROW-MODIFIED	2	The row was modified.
ROW-CREATED	3	The row was created.

- The ROW-STATE function returns the Unknown value (?) when the specified temp-table buffer:
  - Does not contain a record
  - Is an after-image table with no associated before-image table
- You can invoke the ROW-STATE function from within a WHERE clause (unlike the corresponding attribute). For example:

```
WHERE ROW-STATE(ttOrder) = ROW-MODIFIED.
```

See also

Buffer object handle. ROW-STATE attribute, TRACKING-CHANGES attribute

ROWID function

Returns the unique internal identifier of the database record currently associated with the record buffer you name. This internal identifier has the data type `ROWID`, which is supported for OpenEdge and all other DataServer databases.

Note: The `ROWID` function corresponds to the `ROWID` attribute.

This function replaces the `RECID` function for most applications. However, you must use the `RECID` function for maintaining schema objects (file and field relationships) in the ABL meta-schema files.

Syntax

```
ROWID ( record )
```

record

The name of the record whose `ROWID` you want.

To use the ROWID function with a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

Example

You might decide that you do not want to lock a record until the user starts to update that record. In the example procedure, the FIND statement reads a Customer record without locking the record. The ROWID function puts the internal database identifier of that record in the crowid variable. If the user decides to update the CreditLimit field, the procedure finds the record again using the value in crowid. The second FIND statement reads the record again, this time placing an EXCLUSIVE-LOCK on it. Because the record is first found with NO-LOCK, it is possible for the record to be updated by another user after the first FIND and before the second.

r-rowid.p

```
DEFINE VARIABLE response AS LOGICAL NO-UNDO.
DEFINE VARIABLE crowid   AS ROWID   NO-UNDO.

REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum NO-LOCK.
  crowid = ROWID(Customer).
  DISPLAY Customer.Name.
  response = YES.
  UPDATE response LABEL "Update CreditLimit ?".
  IF response THEN DO:
    FIND Customer WHERE ROWID(Customer) = crowid EXCLUSIVE-LOCK.
    UPDATE Customer.CreditLimit.
  END.
END.
```

Notes

- Use the ROWID function to rapidly retrieve a previously identified record, even if that record has no unique index.
- The ROWID data type is a variable-length byte string capable of representing a record identifier for any DataServer database. However, the scope of a specific ROWID returned by the ROWID function depends on the DataServer and possibly the table within a database. The ROWID values for some DataServers change whenever the corresponding record is modified. For others, a ROWID value can change when a particular column in a table is modified. For more information on how different DataServers derive and work with ROWID values, see the OpenEdge DataServer Guides (*Use the Microsoft SQL Data Server* and *Use the Oracle Data Server*).
- You cannot return a ROWID for a view because view records do not have unique identifiers.
- You can compare ROWID values using the ABL relational operators (=, >, <, <>, >=, and <=), such as in the WHERE option of the Record phrase.
- You can use a ROWID value in a REPOSITION statement to specify the new position for a query cursor.
- If you want a called procedure to use the same record as a calling procedure, use the ROWID function to ensure that you are retrieving the same record. Use a SHARED ROWID variable or procedure parameter to communicate the ROWID of a record from one procedure to another. The second procedure can then find the same record. This is an alternative to using shared buffers or buffer parameters.
- You can store a ROWID value in a work table or temp-table, but not directly in a database table.
- You can use the STRING function to convert a ROWID value to a character string, and convert it back to a ROWID value using the TO-ROWID function.

- You do not have to explicitly check to see whether a record is AVAILABLE before using the ROWID function. The ROWID function returns the Unknown value (?) if a record cannot be accessed.

  This example checks the ROWID for each Customer record returned for a query to determine if another record exists to update. If no more records exist, the update loop (QuickFix) terminates.

```
OPEN QUERY qCustomer FOR EACH Customer
  WHERE Customer.Balance > 5000 AND Customer.Balance < 6000.

QuickFix:
REPEAT:
  GET NEXT qCustomer.
  IF ROWID(Customer) = ? THEN LEAVE QuickFix.
  ELSE UPDATE Customer.
END. /* QuickFix */
```

- The ROWID of a record in a partitioned table identifies both the partition and the address of the record in the partition. If a partition field is updated in the record and, as a result, the record is moved to a different partition, the record's ROWID will change.

See also

DEFINE BUFFER statement, DEFINE VARIABLE statement, RECID function, Record phrase, REPOSITION statement, STRING function, TO-ROWID function, ROWID attribute, VAR statement

RUN statement

Calls an ABL procedure. This procedure can be local to or remote from the current session, external from or internal to the current procedure, and either synchronous or asynchronous. When a local or remote procedure is called synchronously, the calling procedure resumes execution only after the called procedure completes execution. When a remote procedure is called asynchronously, the calling procedure resumes execution immediately after the remote request is sent to the application server.

The RUN statement can also call functions or routines that reside in the Windows Dynamic Link Libraries (DLLs) or in UNIX shared libraries. The called routine must first be declared like an ABL internal procedure. The procedure declaration must be in the same file as the RUN statement.

You can also use the RUN statement to create and associate a procedure object with a Web service, and invoke a Web service operation.

Syntax

Use the following syntax to run an external procedure or method:

```
RUN
  { extern-proc-name
    | VALUE ( extern-expression )
    | path-name<<member-name>>
  }
  [ PERSISTENT | SINGLE-RUN | SINGLETON [ SET proc-handle ] ]
  [ ON [ SERVER ] {server-handle | session-handle }
      [ TRANSACTION DISTINCT ]
      [ ASYNCHRONOUS
          [ SET async-request-handle ]
          [ EVENT-PROCEDURE event-internal-procedure
              [ IN procedure-context ] ] |
          [ EVENT-HANDLER event-handler-name
              [ EVENT-HANDLER-CONTEXT event-handler-context ] ]
      ]
  ]
  [ ( parameter [ , parameter ] ... ) ]
  [ argument ] ...
  [ NO-ERROR ]
```

Use the following syntax to run an internal procedure or method:

```
RUN
  { intern-proc-name | VALUE ( intern-expression) }
  [ IN proc-handle ]
  [ ASYNCHRONOUS
      [ SET async-request-handle ]
      [ EVENT-PROCEDURE event-internal-procedure
          [ IN procedure-context ] ] |
      [ EVENT-HANDLER event-handler-name
          [ EVENT-HANDLER-CONTEXT event-handler-context ] ]
  ]
  [ ( parameter [ , parameter ] ... ) ]
  [ NO-ERROR ]
```

Use the following syntax to create and associate a procedure object with a Web service:

```
RUN portTypeName [ SET hPortType ] ON SERVER hWebService [ NO-ERROR ] .
```

Use the following syntax to invoke a Web service operation:

```
RUN operationName IN hPortType
  [ ASYNCHRONOUS
    [ SET async-request-handle ]
    [ EVENT-PROCEDURE event-internal-procedure
      [ IN procedure-context ] ]
    [ EVENT-HANDLER event-handler-name
          [ EVENT-HANDLER-CONTEXT event-handler-context ] ]
  [ ( parameter[ , parameter ]... ) ]
  [ NO-ERROR ].
```

extern-proc-name

The name of the (local or remote) external procedure to run. On UNIX, external procedure names are case sensitive; in Windows, they are not. If you specify a relative pathname, ABL searches the directories (and libraries, on platforms that support libraries) defined in the PROPATH environment variable. With *extern-proc-name*, you can specify a local or remote procedure. In Windows, the pathname cannot contain characters outside of the non-Unicode code page. See *Internationalize ABL Applications* for more information about Unicode and code pages.

VALUE (*extern-expression*)

An expression that returns the name of the (local or remote) external procedure you want to run.In Windows, the pathname cannot contain characters outside of the non-Unicode code page.

path-name<<*member-name*>>

The pathname of an r-code library and the name of an r-code file in that library. To specify an r-code file in a library, you must use the double angle brackets as shown. If you specify a relative library pathname, ABL searches the libraries defined in the PROPATH environment variable. In Windows, the pathname cannot contain characters outside of the non-Unicode code page.

PERSISTENT [SET *proc-handle*]

Specifies that the external procedure be instantiated and run as a persistent procedure.

You can return the handle to the persistent procedure in *proc-handle* specified as a field, variable, class property, or output parameter defined with the HANDLE data type. If you do not specify *proc-handle*, you can find the procedure handle for this procedure using the FIRST-PROCEDURE or LAST-PROCEDURE attributes of the SESSION system handle.

For remote persistent procedures, *proc-handle* returns the proxy (remote) handle to the persistent procedure. You can also find the proxy handle by using the FIRST-PROCEDURE or LAST-PROCEDURE attributes of the server object handle for the application server specified with the ON SERVER option.

A persistent procedure creates and maintains its context after it returns to the caller. Other external procedures can access this context through procedure triggers and internal procedures or user-defined functions defined in the persistent procedure. A RUN statement that runs and creates a persistent procedure context is referred to as an *instantiating RUN statement*. Thus, a persistent procedure has the following characteristics:

- The procedure does not go out of scope when it returns: its context and most of its allocated resources remain active, including input parameters, widgets, variables, buffers, temp-tables,

work tables, and triggers created during procedure execution. However, all **static** dialog boxes, their child widgets, and related triggers created during its execution are destroyed when the procedure returns to the caller. This makes all other windows and dialog boxes in the application available for input.

- All buffers passed as parameters to a persistent procedure are treated as local buffers in the persistent context. When the procedure instantiation returns, the output value of the buffer parameter is returned, as usual, to the calling procedure. However, any cursor positioning established during execution of the instantiating RUN statement is lost to the persistent context once the procedure returns; the AVM creates a copy of the buffer parameter and resets its cursors as an initially defined local buffer.
- If the procedure obtains any schema share locks (through database access) while executing, these remain in effect after the procedure returns, until the procedure is deleted.
- Each time you run a procedure persistently, you create a new instance of its procedure context. All of its data, buffers, and widgets are duplicated and separately managed by the new instantiation until the procedure instance is deleted.

Note:

If you run an application that creates persistent procedures from an ADE tool (for example, the Procedure Editor or AppBuilder), that tool removes all instances of persistent procedures still created when the application terminates.

Transaction scoping is the same whether you run a procedure as persistent or not. Any transaction which begins inside a persistent procedure is scoped to the block that starts the transaction.

The order of the PERSISTENT option and the ON SERVER option is interchangeable.

You can use PERSIST as an abbreviation for PERSISTENT.

If you want to invoke persistent procedures on a session-free Progress Application Server (PAS) for OpenEdge server, use either SINGLE-RUN or SINGLETON as an alternative to the PERSISTENT option.

SINGLE-RUN [SET *proc-handle*]

Initializes the specified external procedure to run on an application server or on the client (by using a SESSION handle) in single-run mode, but it does not initially run the procedure.

The SINGLE-RUN option is particularly useful for a session-free PAS for OpenEdge instance because it allows you to call internal procedures or user-defined functions in a remote external procedure without binding the client to a single application server session. In addition, network traffic is reduced when you run a procedure as a single-run.

In single-run mode, the external procedure is instantiated only when an internal procedure or user-defined function that it defines is invoked remotely. The single-run procedure is deleted when the internal procedure or user-defined function completes. Compare with the SINGLETON option, where the singleton procedure remains instantiated after the internal procedure or user-defined function completes. See the description under *intern-proc-name* for more information on how internal procedures and user-defined functions run in SINGLE-RUN mode.

You can return the proxy handle to the single-run procedure in *proc-handle* specified as a field, variable, class property, or output parameter defined with the HANDLE data type. If you do not specify *proc-handle*, you can find the proxy handle for this procedure using the FIRST-PROCEDURE and LAST-PROCEDURE attributes of the server object handle for the application server specified using the ON SERVER option.

Each time you call an internal procedure or user-defined function on a single-run procedure, you create a new instance of its procedure context, which only exists during the execution of the internal procedure or user-defined function. All of its data, buffers, and widgets are duplicated and separately managed by the new instantiation until the procedure instance is deleted.

Note:

You must use the ON SERVER option when you use the SINGLE-RUN option.

You cannot use SINGLE-RUN with external procedures that have parameters.

If you attempt to delete a single-run procedure from a client, you only delete the proxy handle. A single-run procedure remains instantiated for an application server session only until its instantiating internal procedure or user-defined function completes execution.

When a single-run procedure is invoked and the procedure is already instantiated as a singleton, the application server or client deletes the singleton procedure. The procedure is re-instantiated as a single-run. When the execution of the internal procedure or user-defined function completes, the application server or client deletes the single-run procedure but does not re-instantiate the procedure as a singleton.

You cannot use the ASYNCHRONOUS option with SINGLE-RUN when instantiating an external procedure. However you can run an internal procedure of a SINGLE-RUN procedure asynchronously using the following syntax: RUN *intern_proc-name* IN *proc-handle*. . . ASYNCHRONOUS.

You can use the EXPORT( ) method on the SESSION handle to add a procedure that will be run in single-run mode to the list of authorized remote procedures on the application server.

For single-run procedures, the main block does not run until the internal procedure or user-defined function first runs. Therefore, an ERROR or STOP condition occurring in the main block is raised during the first execution of the internal procedure or user-defined function.

SINGLETON [SET *proc-handle*]

Initializes the specified external procedure to run on an application server or on the client (by using a SESSION handle) in singleton mode, but it does not initially run the procedure.

The SINGLETON option is particularly useful for a session-free application server session because it allows you to call internal procedures or user-defined functions in a remote external procedure without binding the client to a single application server session. In addition, network traffic is reduced when you run a procedure as a singleton.

In singleton mode, the external procedure is instantiated only when an internal procedure or user-defined-function that it defines is invoked remotely, and only if the singleton procedure is not already instantiated for a previous internal remote invocation.

The external procedure remains instantiated after its internal procedure or user-defined function completes execution. Compare with the single-run mode, where the procedure is deleted after an internal procedure or user-defined function completes execution. With the SINGLETON option, the external procedure remains instantiated and is used the next time one of its internal procedures or user-defined functions is run. See the description under *intern-proc-name* for more information on how internal procedures and user-defined functions run in singleton mode.

You can return the proxy handle to a singleton procedure in *proc-handle* specified as a field, variable, class property, or output parameter defined with the HANDLE data type. If you do not specify *proc-handle*, you can find the proxy handle for this procedure using the FIRST-PROCEDURE and LAST-PROCEDURE attributes of the server object handle for the application server specified using the ON SERVER option.

An external procedure called with the SINGLETON option runs in the same way as other procedures, but with these differences:

- The procedure does not go out of scope when it returns. Most of its allocated resources remain active, including objects, variables, buffers, temp-tables, work tables, and triggers created during procedure execution. However, you should not keep or rely on any context information maintained for a singleton procedure. Each application server session maintains its own context information independent of (and probably differing from) other application server sessions. Since there is no guarantee which application server session will process a given request, there is no way of knowing what context information will be available.
- If the procedure obtains any schema share locks (through database access) while executing, these remain in effect after the procedure returns, until the procedure is deleted.

Note:

You must use the ON SERVER option when you use the SINGLETON option.

You cannot use the SINGLETON option with external procedures that have parameters.

Only one instance of a given singleton procedure exists for a given application server session, no matter how many clients initialize the procedure with the SINGLETON option. Therefore, all clients access the same instance of a singleton procedure in requests to the same application server session.

If you attempt to delete a singleton procedure from a client using the DELETE PROCEDURE statement, you only delete the proxy handle. A singleton remains instantiated for an application server session until session code deletes it or the session terminates.

When a single-run procedure is invoked and the procedure is already instantiated as a singleton, the application server or client deletes the singleton procedure. The procedure is re-instantiated as a single-run. When the execution of the internal procedure or user-defined function completes, the application server or client deletes the single-run procedure but does not re-instantiate the procedure as a singleton.

You cannot use the ASYNCHRONOUS option with SINGLETON when instantiating an external procedure. However you can run an internal procedure of a SINGLETON procedure asynchronously using the following syntax: RUN *intern-proc-name* IN *proc-handle*. . . ASYNCHRONOUS.

You can use the EXPORT( ) method on the SESSION handle to add a procedure that will be run as a singleton to the list of authorized remote procedures on the application server.

For singleton procedures, the main block does not run until the internal procedure or user-defined function first runs. Therefore, an ERROR or STOP condition occurring in the main block is raised during the first execution of the internal procedure or user-defined function.

ON [SERVER] *server-handle*

Tells the AVM to run the procedure remotely in the application server that the HANDLE variable, *server-handle* , refers to.

With the ASYNCHRONOUS option, *server-handle* causes the called procedure to run **asynchronously** in the remote session. Control returns immediately to the statement following the RUN statement. Execution of any specified event procedure or method occurs in the context of an input-blocking or PROCESS EVENTS statement.

The order of any PERSISTENT, SINGLE-RUN, or SINGLETON option and the ON SERVER option is interchangeable.

ON [SERVER] *session-handle*

Tells the AVM to run the procedure locally in the current ABL session, specified by the value of the SESSION system handle (*session-handle*).

With the ASYNCHRONOUS option, *session-handle* causes the called procedure to run **synchronously** in the local session, followed immediately by execution of any specified *event-internal-procedure*. Only after execution of the specified *event-internal-procedure* does control return to the statement following the RUN statement.

Note: This order of execution is different than for a remote procedure call using the *server-handle*.

The order of the PERSISTENT, SINGLE-RUN, or SINGLETON option and the ON SERVER option is interchangeable.

TRANSACTION DISTINCT

Tells the AVM not to propagate the calling procedure's transaction to the application server. Although the current version of ABL does not allow transaction propagation, future versions might. Thus, to accommodate this possibility without breaking current code, the current version of ABL allows you to specify this option with *server-handle*.

Note: It is an error to specify TRANSACTION DISTINCT with a *session-handle*.

ASYNCHRONOUS [SET *async-request-handle*]

Specifies that the remote procedure is to be called as an asynchronous request. By default, the remote procedure is called synchronously. The handle to the asynchronous request is returned in *async-request-handle*, which must be a field, variable, or parameter defined with the HANDLE data type. If you specify ASYNCHRONOUS but do not specify SET *async-request-handle*, you can find the handle for the asynchronous request using the LAST-ASYNC-REQUEST attribute of the *server-handle* specified by the ON option. You can also locate the asynchronous request handle by walking the chain between the FIRST-ASYNC-REQUEST and LAST-ASYNC-REQUEST attributes of *server-handle*, searching on the PROCEDURE-NAME attribute of each request handle.

For a Web service operation invoked asynchronously, the handle that is set to the asynchronous request object created for the asynchronous request.

Note: You cannot use the ASYNCHRONOUS option with either the SINGLE-RUN or SINGLETON option when instantiating an external procedure. However you can run an internal procedure of a SINGLETON procedure asynchronously using the following syntax: RUN *internal_procedure*IN*proc-handle*. . . ASYNCHRONOUS.

EVENT-PROCEDURE *event-internal-procedure*

Specifies a quoted string or character expression representing the name of an internal procedure that resides within *procedure-context*. When the response from the asynchronous request is received (that is, a PROCEDURE-COMPLETE event occurs), the specified internal procedure is called during subsequent execution of a PROCESS EVENTS or input-blocking statement (such as WAIT-FOR). The specified *event-internal-procedure* processes any parameters and errors returned from the asynchronous request. If not specified, no event procedure is executed when the PROCEDURE-COMPLETE event occurs for the asynchronous request.

For information on how the *event-internal-procedure* handles parameters from the asynchronous request, see the *parameter* option. For information on how the *event-internal-procedure* handles errors from the asynchronous request, see the NO-ERROR option.

Note: The EVENT-PROCEDURE and IN keyword syntax are now replaced by the EVENT-HANDLER and the EVENT-HANDLER-CONTEXT keywords. Progress recommends that you use the new syntax that supports both procedures and methods as callback for ASYNCHRONOUS RUN..

IN *procedure-context*

A handle to an active procedure that contains the internal procedure specified by *event-internal-procedure*. If not specified, THIS-PROCEDURE is used as the *procedure-context* value.

Note: The EVENT-PROCEDURE and IN keyword syntax are now replaced by the EVENT-HANDLER and the EVENT-HANDLER-CONTEXT keywords. Progress recommends that you use the new syntax that supports both procedures and methods as callback for ASYNCHRONOUS RUN.

EVENT-HANDLER *event-handler-name*

A character expression that results in the name of an internal procedure or a method that is specified as a callback handler. When the response from the asynchronous request is received (that is, a PROCEDURE-COMPLETE event occurs), the specified *event-handler-name* (procedure or method) is called during subsequent execution of PROCESS EVENTS or input-blocking statement (such as WAIT-FOR). The specified event handler procedure or method processes any parameters and errors returned from the asynchronous request. If not specified, no event procedure or method is executed when the PROCEDURE-COMPLETE event occurs for the asynchronous request. For information on how the event procedure or method handles parameters from the asynchronous request, see the *parameter* option. For information on how the event handler (procedure or method) handles errors from the asynchronous request, see the NO-ERROR option.

When using a method as a callback handler, you can overload the method and the AVM will identify the most appropriate method to run by using the the signature of the method. The signature of the method is determined based on the output and input-output parameters passed to the asynchronous procedure executed by the RUN statement.

EVENT-HANDLER-CONTEXT *event-handler-context*

The *event-handler-context* value resolves to either a handle to an active procedure or the object-oriented ABL object containing the method specified by *event-handler-name*.

- If the value of *event-handler-context* is an object-oriented ABL object, then the *event-handler-name* specified must be a public method.
- The value of *event-handler-context* may optionally be THIS-PROCEDURE when the RUN statement is invoked from within a runnable block of a procedure.
- The value of *event-handler-context* may optionally be THIS-OBJECT when the RUN statement is invoked from within a runnable block of an object-oriented ABL object.

(*parameter* **[** , *parameter* **]** **...**)

Specifies one or more parameters to pass to the called procedure or method.

For the parameter passing syntax, see the Parameter passing syntax reference entry.

Parameters must be defined in the called procedure or method. (See the DEFINE PARAMETER statement reference entry.) They must be passed in the same order as they are defined, and they must have compatible data types. The AVM attempts to convert values for data types that do not match. If the AVM cannot convert the value for a mismatched data type, the RUN statement fails with an error condition.

For OUTPUT parameters of an asynchronous remote procedure call only, you can specify *parameter-name* AS *primitive-type-name* as a prototype. The *parameter-name* is an arbitrary place-holder name and *primitive-type-name* must specify the ABL data type of the corresponding OUTPUT parameter in the asynchronous remote procedure. You can also specify OUTPUT parameters for an asynchronous remote procedure or method using a local *field*, *variable*, TABLE *temp-table-name*, TABLE-HANDLE *temp-table-handle*, DATASET *dataset-name*, or DATASET-HANDLE *dataset-handle*. However, note that the asynchronous remote procedure or method does not return any values to OUTPUT or INPUT-OUTPUT parameters on the RUN statement. These parameters are place holders only for values returned by the remote procedure to the specified *event-handler-name* procedure or method.

Any specified *event-handler-name* procedure or method can define only INPUT parameters and must define one INPUT parameter for each OUTPUT or INPUT-OUTPUT parameter defined in the asynchronous remote procedure. Each INPUT parameter for the *event-handler-name* procedure or method, must match the corresponding remote procedure OUTPUT or INPUT-OUTPUT parameter in order and data type. (As with other procedures or methods, the AVM attempts to convert the values for data types that do not match.) The asynchronous remote procedure returns the values of these parameters to the INPUT parameters of the *event-handler-name* procedure or method after the remote procedure completes execution and the client session processes the associated PROCEDURE-COMPLETE event.

If you are running a *event-handler-name* procedure or method declared as a Windows dynamic link library (DLL) or UNIX shared library routine, you must match any RETURN parameter specified by a DEFINE PARAMETER statement with a corresponding OUTPUT parameter in the RUN statement. If the internal procedure does not specify the RETURN parameter, do not specify the corresponding OUTPUT parameter in the RUN statement.

For external procedures, the parenthesized list of run-time parameters must precede any compile-time arguments.

argument

A constant, field name, variable name, or expression that you want to pass as a compile-time argument to the external procedure you are running.

When you pass arguments to an external procedure, the AVM converts those arguments to character format. ABL recompiles the called procedure, substitutes arguments, and then runs the procedure. You cannot precompile a procedure to which you pass arguments. (If you use shared variables instead of arguments, the procedure can be precompiled. This yields more efficient code.)

Note: You cannot pass compile-time arguments in a call to an internal procedure.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages. This does **not** mean that all errors produced by the called procedure are suppressed; only errors caused by the RUN statement itself.

If a specified local or synchronous remote procedure throws an error back to the RUN statement, you can check the ERROR-STATUS system handle for information on errors that occurred.

For an asynchronous remote procedure, the result depends on where the errors occur. If the errors occur during the send phase of the asynchronous request, this raises the ERROR condition on the RUN statement in the client which will be handled by the NO-ERROR option. If the errors occur during execution of the remote request and are returned by the application server, the RUN statement completes successfully and you must check for ERROR or STOP conditions in the specified *event-handler-name* (procedure or method) for the PROCEDURE-COMPLETE event. In the event handler, you check the attributes of the *async-request-handle*.

If the specified procedure is not found or if an attempted compile of the same fails, the AVM raises the STOP condition (for local procedures) or ERROR condition (for remote procedures). So only in the remote case can NO-ERROR be used to handle these conditions.

intern-proc-name

The name of the (local or remote) internal procedure you want to run. The procedure must be declared in the same procedure file as the RUN statement that calls it unless you specify the IN *proc-handle* option or use a super procedure.

If you do not specify the IN *proc-handle* option and there is no internal procedure declared by the specified name, the AVM tries to run an external procedure with the specified name. If the internal procedure is remote, you must specify the IN *proc-handle* option to identify the remote persistent, single-run, or singleton procedure that defines the internal procedure on an application server.

If an internal procedure or user-defined function is run using the *proc-handle* associated with a single-run procedure, the application server session or client:

- Instantiates the single-run procedure.
- Runs the main block.
- Runs the internal procedure or user-defined function.
- Returns any output parameters or return values to the caller.
- Deletes the single-run procedure.

If an internal procedure or user-defined function is run using the *proc-handle* associated with a singleton procedure, the application server session or client:

- Instantiates the singleton procedure if it is not already instantiated by a previous client singleton request.
- Runs the main block.
- Runs the internal procedure or user-defined function.
- Returns any output parameters or return values to the caller.
- The singleton procedure remains instantiated.

Note: For more information on running remote user-defined functions, see the FUNCTION statement section.

VALUE (*intern-expression*)

An expression that evaluates to the name of the internal procedure you want to run.

IN *proc-handle*

Specifies the handle of the external procedure that declares the internal procedure you want to run. You can specify *proc-handle* as a field, variable, parameter, or expression that specifies a valid procedure handle or proxy (remote) persistent procedure handle.

portTypeName

The name of a Web service PortType as specified in the WSDL file.

hPortType

A handle to a procedure object that encapsulates a Web service operation.

hWebService

A handle to the server object bound to the Web service.

operationName

The name of a Web service operation specified in a WSDL file.

Examples

The following procedure displays a simple menu. The user's selection is stored in the selection variable. The INDEX function returns an integer value that indicates the position of the user's selection in a string of characters ("12345"). If the value in the selection variable is not in the list of values, the INDEX function returns a 0. The VALIDATE statement ensures that the INDEX function did not return a zero. If it did, VALIDATE displays the message, "Not a valid choice".

r-run.p

```
DEFINE VARIABLE selection AS CHARACTER NO-UNDO FORMAT "x(1)"
  LABEL "Enter Program Choice".
DEFINE VARIABLE programs  AS CHARACTER NO-UNDO FORMAT "x(15)" EXTENT 5.

/* Create the procedures custrpt.p, custedit.p, ordrpt.p, and ordedit.p.*/
ASSIGN
  programs[1] = "custrpt.p"
  programs[2] = "custedit.p"
  programs[3] = "ordrpt.p"
  programs[4] = "ordedit.p"
  programs[5] = "r-exit.p".

REPEAT:
  FORM HEADER TODAY "MASTER MENU" AT 35 STRING(TIME,"hh:mm") to 79.
  FORM SKIP(3)
   "1 - Customer Listing"  AT 30
   "2 - Customer Update" AT 30
   "3 - Order Listing"  AT 30
   "4 - Order Update"   AT 30
   "5 - Quit System"    AT 30
   selection COLON 28 AUTO-RETURN WITH SIDE-LABELS NO-BOX 1 DOWN.

UPDATE selection
    VALIDATE(INDEX("12345",selection) NE 0, "Not a valid choice").
  HIDE ALL.
  RUN VALUE(programs[INDEX("12345",selection)]).
END.
```

In the RUN statement, the INDEX function returns the position of the user's selection in a character string. Suppose you chose option 2 from the menu. That option occupies the second position in the "12345" character string. Therefore, the INDEX function returns the number two (2). Using this number, the RUN statement reads, RUN VALUE(programs[2]). According to the assignments at the top of the procedure, the value of programs[2] is custedit.p. Now the RUN statement reads, RUN custedit.p, and the r-run.p procedure runs the custedit.p procedure.

The following two external procedures, `r-runper.p` and `r-perprc.p`, illustrate the `PERSISTENT` and `IN` *`proc-handle`* options of the `RUN` statement. The first procedure, a non-persistent control procedure, sets up a window to run and manage the second procedure as a persistent procedure.

r-runper.p

```
DEFINE VARIABLE phand AS HANDLE NO-UNDO.
DEFINE VARIABLE nhand AS HANDLE NO-UNDO.
DEFINE VARIABLE whand AS HANDLE NO-UNDO.

DEFINE BUTTON bStart LABEL "Start Customer Query".
DEFINE BUTTON bRecall LABEL "Recall All Hidden Queries".
DEFINE BUTTON bExit LABEL "Exit".

DEFINE FRAME ControlFrame SKIP(.5) SPACE(2)
  bStart bRecall bExit SPACE(2) SKIP(.5).

ON CHOOSE OF bStart IN FRAME ControlFrame RUN r-perprc.p PERSISTENT.

ON CHOOSE OF bRecall IN FRAME ControlFrame DO:
  phand = SESSION:FIRST-PROCEDURE.
  DO WHILE VALID-HANDLE(phand):
    IF phand:PRIVATE-DATA = "Customer Browse" THEN
      RUN recall-query IN phand.
    phand = phand:NEXT-SIBLING.
  END.
END.

ON CHOOSE OF bExit IN FRAME ControlFrame DO:
  phand = SESSION:FIRST-PROCEDURE.
  DO WHILE VALID-HANDLE(phand):
    nhand = phand:NEXT-SIBLING.
    IF phand:PRIVATE-DATA = "Customer Browse" THEN
      RUN destroy-query IN phand.
      phand = nhand.
  END.
  APPLY "RETURN" TO THIS-PROCEDURE.
END.

SESSION:SYSTEM-ALERT-BOXES = TRUE.
CREATE WINDOW whand ASSIGN
  TITLE            = "Customer Query Control"
  SCROLL-BARS      = FALSE
  MESSAGE-AREA     = FALSE
  MAX-HEIGHT-CHARS = FRAME ControlFrame:HEIGHT-CHARS
  MAX-WIDTH-CHARS  = FRAME ControlFrame:WIDTH-CHARS.

CURRENT-WINDOW = whand.
ENABLE ALL WITH FRAME ControlFrame.
WAIT-FOR RETURN OF THIS-PROCEDURE.
```

r-perprc.p

```
DEFINE VARIABLE custwin AS HANDLE NO-UNDO.

DEFINE BUTTON bName    LABEL "Query on Name".
DEFINE BUTTON bBalance LABEL "Query on Balance".
DEFINE BUTTON bCredit  LABEL "Query on Credit".
DEFINE BUTTON bHide    LABEL "Hide Query".
DEFINE BUTTON bCancel  LABEL "Cancel".

DEFINE QUERY custq FOR Customer.
DEFINE BROWSE custb QUERY custq
  DISPLAY Customer.Name Customer.Balance Customer.CreditLimit Customer.Phone
```

```
    WITH 10 DOWN.

DEFINE FRAME CustFrame custb SKIP
    bName bBalance bCredit bHide bCancel.

ON CHOOSE OF bName IN FRAME CustFrame DO:
  custwin:TITLE = "Customers by Name".
  OPEN QUERY custq FOR EACH Customer BY Customer.Name.
END.

ON CHOOSE OF bBalance IN FRAME CustFrame DO:
  custwin:TITLE = "Customers by Balance".
  OPEN QUERY custq FOR EACH Customer BY Customer.Balance DESCENDING.
END.

ON CHOOSE OF bCredit IN FRAME CustFrame DO:
  custwin:TITLE = "Customers by Credit".
  OPEN QUERY custq FOR EACH Customer BY Customer.CreditLimit DESCENDING.
END.

ON VALUE-CHANGED OF BROWSE custb DO:
  IF Customer.Balance >= (Customer.CreditLimit * 0.75) THEN DO:
    BELL.
    MESSAGE "Evaluate" Customer.Name "for credit increase.".
  END.
END.

IF THIS-PROCEDURE:PERSISTENT THEN DO:
  THIS-PROCEDURE:PRIVATE-DATA = "Customer Browse".
  CREATE WIDGET-POOL.
END.

CREATE WINDOW custwin ASSIGN
  TITLE            = "Customer Browser"
  SCROLL-BARS      = FALSE
  MAX-HEIGHT-CHARS = FRAME CustFrame:HEIGHT-CHARS
  MAX-WIDTH-CHARS  = FRAME CustFrame:WIDTH-CHARS.

THIS-PROCEDURE:CURRENT-WINDOW = custwin.

ENABLE ALL WITH FRAME CustFrame.

IF THIS-PROCEDURE:PERSISTENT THEN DO:
  ON CHOOSE OF bCancel IN FRAME CustFrame DO:
    RUN destroy-query.
  END.
  ON CHOOSE OF bHide IN FRAME CustFrame DO:
    custwin:VISIBLE = FALSE.
  END.
END.
ELSE
  WAIT-FOR CHOOSE OF bHide, bCancel IN FRAME CustFrame.

PROCEDURE recall-query:
  custwin:VISIBLE = TRUE.
END PROCEDURE.

PROCEDURE destroy-query:
  DELETE PROCEDURE THIS-PROCEDURE NO-ERROR.
  DELETE WIDGET-POOL.
END PROCEDURE.
```

The control procedure, r-runper.p, runs r-perprc.p each time you choose the Start Customer Query button. Each time it runs, r-perprc.p creates (instantiates) an additional context instance for the persistent procedure, including an additional window to open customer queries. When you choose the Recall All Hidden Queries button from the control window, r-runper.p calls the recall-query internal procedure in each instance of r-perprc.p to redisplay its window. Similarly, when you choose the Exit button, r-runper.p calls the destroy-query internal procedure in each instance of r-perprc.p to delete its context instance; r-runper.p then applies the RETURN event to itself to terminate by completing the WAIT-FOR statement.

The r-perprc.p procedure sets up a customer query that you can re-open three different ways: by name, by balance, or by credit. Each instance of r-perprc.p maintains a separate query for its own local customer buffer. Note that by testing and setting attributes of the THIS-PROCEDURE system handle, r-perprc.p can run either persistently or non-persistently. The basic difference is how the procedure maintains its own context. For example, when running persistently, it defines a trigger on the bCancel button to run its own deletion procedure, destroy-query, to terminate; when running non-persistently, it completes a WAIT-FOR statement with the bCancel button to terminate.

The following example shows how you might implement an asynchronous request. The procedure r-async.p runs persistently from a user-interface trigger, perhaps in response to a menu choice. This procedure, in turn, sends a request to run runReport.p on an application server, which provides an inventory report for the specified date.

When r-async.p returns, the user-interface trigger ends and the application returns to its WAIT-FOR state. The user continues to use the application in the normal way while the inventory report runs on the application server.

When runReport.p finishes running, a PROCEDURE-COMPLETE event occurs. This event causes the internal procedure reportDone to run automatically within the context of the application's WAIT-FOR statement. Whatever the user is doing in the application, reportDone displays an alert box indicating whether or not the inventory report completed successfully and the number of lines (numLines) that were output for the report. (The bold ABL statements indicate the code required to support asynchronous requests to run runReport.p.)

```
DEFINE INPUT PARAMETER invDate AS DATE NO-UNDO.
DEFINE VARIABLE sh AS HANDLE NO-UNDO. /* Server handle */
DEFINE VARIABLE ah AS HANDLE NO-UNDO. /* Asynchronous request handle */

CREATE SERVER sh.
sh:CONNECT("-URL http://slater:8810/inventory/apsv  -sessionModel Session-free").
RUN runReport.p ON SERVER sh
  ASYNCHRONOUS SET ah EVENT-HANDLER "reportDone" EVENT-HANDLER-CONTEXT THIS-PROCEDURE
  (invDate, OUTPUT numLines AS INTEGER).
RETURN.

PROCEDURE reportDone:
  DEFINE INPUT PARAMETER numLines AS INTEGER.

  IF ah:ERROR OR ah:STOP THEN
    MESSAGE "An error occurred when running your" SKIP
      "Inventory report for" invDate "." SKIP
      "The error is:" ERROR-STATUS:GET-MESSAGE(1)
      VIEW-AS ALERT-BOX.
  ELSE
    MESSAGE "Your Inventory report for " invDate SKIP
      "has completed successfully." SKIP
      numLines "report lines were generated"
      VIEW-AS ALERT-BOX.

sh:DISCONNECT().
  DELETE OBJECT sh.
  DELETE OBJECT THIS-PROCEDURE. /* Persistent proc no longer needed */
END PROCEDURE.
```

This example shows an asynchronous call that potentially returns an error object through the ERROR-OBJECT attribute of the Asynchronous request object handle. When server-error.p completes, procDone checks the value of ERROR, and if true, it assigns the instance referenced by ERROR-OBJECT to myAppErrorObj via a cast and then displays the information available, including the attributes inherited from Progress.Lang.Error and an ErrorCode attribute from the user-defined UserDefinedAppError class.

r-client-async-error.p

```
ROUTINE-LEVEL ON ERROR UNDO, THROW.

DEFINE VARIABLE hRequest AS HANDLE NO-UNDO.
DEFINE VARIABLE hServer AS HANDLE NO-UNDO.
DEFINE VARIABLE lReturn AS LOGICAL NO-UNDO.

CREATE SERVER hServer.

lReturn = hServer:CONNECT("-pfcspair1.npf").

IF NOT hServer:CONNECTED( ) THEN DO:
  MESSAGE "Failed to connect to application server".
  RETURN.
END.

RUN server-error.p ON SERVER hServer
  ASYNCHRONOUS SET hRequest EVENT-HANDLER "procDone" EVENT-HANDLER-CONTEXT THIS-PROCEDURE
 (OUTPUT numLines AS INTEGER).

WAIT-FOR PROCEDURE-COMPLETE OF hRequest.

DELETE OBJECT hRequest NO-ERROR.
hServer:DISCONNECT().
DELETE OBJECT hServer NO-ERROR.

PROCEDURE procDone:
  DEFINE INPUT PARAMETER numLines AS INTEGER.
  DEFINE VAR myErrObj AS Progress.Lang.Error.
  DEFINE VAR myAppErrObj AS UserDefinedAppErr.
  DEFINE VAR iTmp AS INTEGER NO-UNDO.

  IF hRequest:ERROR THEN DO:
    MESSAGE "An error occurred when running server-error.p".

    myErrObj = SELF:ERROR-OBJECT.
    myAppErrObj = CAST (myErrObj, UserDefinedAppErr).

    MESSAGE "error object NumMessages: " myErrObj:NumMessages SKIP
      "Severity: " myAppErrObj:Severity SKIP
      "CallStack: " myAppErrObj:CallStack SKIP
      "Error Code: " myAppErrObj:ErrorCode.
      REPEAT iTmp = 1 TO myAppErrObj:NumMessages:
        MESSAGE "ErrorMsg(" iTmp "): " myAppErrObj:GetMessage(iTmp) SKIP
          "ErrorNum(" iTmp "): " myAppErrObj:GetMessageNum(iTmp).
      END.
    END.
  ELSE IF NOT hRequest:STOP THEN
      MESSAGE "server-error.p was has completed successfully." SKIP
        numLines " was output parameter".
END PROCEDURE.
```

The example r-singlerun.p illustrates how to run any internal procedure in aproc.p as a single-run. The application server does not become bound to the client and network traffic is minimized when you run a procedure as single-run.

When the client executes the RUN aproc.p statement, a proxy handle to the single-run procedure on the application server is set without sending a message to the application server. When the client sends the request to run internalProcA, the application server runs the main block of aproc.p and then runs the internal procedure internalProcA. After internalProcA completes and ends, the application server deletes the single-run procedure. The final line deletes the proxy handle on the client without sending a message to the application server.

r-singlerun.p

```
RUN aproc.p SINGLE-RUN SET procHandle ON SERVER serverHandle.
RUN internalProcA IN procHandle (INPUT argIn1, OUTPUT argout1).
DELETE PROCEDURE procHandle.
```

Compare r-singlerun.p with r-singleton.p. In both cases, the application server does not become bound to the client and network traffic is minimized. However, after internalProcA runs in r-singleton.p, the singleton procedure aproc.p is not deleted. It remains instantiated and is available when the call is made to run internalProcB.

r-singleton.p

```
RUN aproc.p SINGLETON SET procHandle ON SERVER serverHandle.
RUN internalProcA IN procHandle (INPUT argIn1, OUTPUT argout1).
RUN internalProcB IN procHandle (INPUT argIn1, OUTPUT argout1).
DELETE PROCEDURE procHandle.
```

Notes

- When you run a procedure and do not indicate that the procedure is external, the AVM first looks for an internal procedure with the specified name. You indicate that a procedure is external in a RUN statement by specifying one or more of these options: *path-name*<<*member-name*>>, PERSISTENT, ON SERVER, SINGLE-RUN, SINGLETON. If no internal procedure is found, or if you indicated the procedure is external, the AVM (local or remote) searches all the directories and libraries in its PROPATH for a usable r-code file of the same name. If the AVM cannot find an r-code file and there is a source code file of the same name, the AVM attempts to compile and run the source code file. Note that all searches for procedure names are not case sensitive.
- ABL procedures can be run recursively (a procedure can run itself).
- When a RUN statement raises the STOP condition, the AVM displays the resulting messages on the current output device, even if you specify NO-ERROR. The AVM also writes these messages to the ERROR-STATUS system handle, but sets ERROR-STATUS:ERROR to FALSE.
- You can run an internal procedure that is declared in the current external procedure or in the procedure you specify with the IN *proc-handle* option. The procedure handle specified by the IN *proc-handle* option can specify either a valid persistent procedure instance or an external procedure that is active on the procedure call stack. The handle can also specify the current external procedure using the THIS-PROCEDURE system handle. You can check the validity of any procedure handle using the VALID-HANDLE function.
- A called external procedure uses any arguments passed to it from the calling procedure by referring to those arguments as numbers enclosed in braces { }. The first argument is {1}, the next is {2}, etc. Any arguments the called procedure does not use are ignored, and any missing arguments are treated as null values. (Note that the null is a legal value in a WHERE or WITH clause, but its occurrence can cause an error at other points in a called procedure.)
- To run an r-code file stored in a library that is not on PROPATH, you must specify the name of the library and the name of the r-code file in the library. Specify these names in the form

path-name<<*member-name*>>, where *path-name* is the pathname of the library and *member-name* is the name of the r-code file. For example, if you have an r-code file called appmenu.r in a library whose pathname is /usr/foo/app.pl, you use this command to run it:

```
RUN /usr/foo/app.pl<<appmenu.r>>.
```

- When running an external procedure, it is good practice to specify the name of the source file in the RUN statement. For example, to run r-exit.p you specify the following:

```
RUN r-exit.p
```

  When you specify a suffix or file extension (such as .p), the AVM first tries replacing that suffix or extension with .r and searches the first directory on your PROPATH for a file with that name. If the r-code file is not found, then it reverts to the original suffix and searches for a source file with that name. If the source file is not found in the first PROPATH directory, then the AVM searches for an r-code file and then a source file in each subsequent directory on your PROPATH until a file is found.

  If you specify the .r suffix in the RUN statement, then the AVM searches only for an r-code file in each directory on your PROPATH. If you omit the extension, then the AVM first adds a .r to the name you specify and searches the first directory for an r-code file with that name. If none is found, then the AVM searches for a source file with no suffix or extension.

- You cannot run an internal procedure with the PERSISTENT, SINGLE-RUN, or SINGLETON options.
- If you run a persistent, single-run, or singleton procedure and a STOP, QUIT, or a RETURN ERROR occurs during execution of the procedure, the procedure is deleted.
- All shared variables, buffers, temp-tables, ProDataSet objects, work tables, and queries remain in scope as long as a persistent procedure instance remains that accesses them. This is true even if the procedure (persistent or non-persistent) that originally defined the shared data has gone out of scope. Shared data can go out of scope only when no persistent procedure remains that references it.
- You cannot run a persistent, single-run, or singleton procedure in which you have defined shared streams or shared frame, browse, or menu widgets. Doing so causes the AVM to raise ERROR on the RUN statement.
- You can remove an instance of a persistent or single-run procedure that you run using the PERSISTENT option using the DELETE PROCEDURE or DELETE OBJECT statement. When you delete the procedure instance, its context goes out of scope and all allocated resources are returned to the system. If the procedure has shared dependencies on the call stack, the delete pends until the dependencies are cleared.
- To run a Windows DLL routine as an internal procedure, you must reference the DLL in a PROCEDURE statement and define its parameters in the associated internal procedure block. For more information on accessing DLL routines from ABL, see *OpenEdge Programming Interfaces*.
- To run a UNIX shared library routine as an internal procedure, you must reference the UNIX shared library in a PROCEDURE statement and define its parameters in the associated internal procedure block. You can declare an internal procedure as a routine in a UNIX shared library in the same manner as declaring a DLL routine. The one exception is that the ORDINAL option is not applicable to UNIX and will be ignored. For example:

```
RUN atoi (INPUT in-string, OUTPUT out-int).
```

- You can define triggers on procedure handles (procedure triggers). You can apply events to any procedure trigger defined either within a persistent procedure or within any external procedure that is active on the procedure call stack. For example:

```
DEFINE VARIABLE phand AS HANDLE NO-UNDO.

RUN persproc.p PERSISTENT SET phand.
. . .
APPLY "RETURN" TO phand.
```

  This code fragment assumes that a trigger is defined within `persproc.p` for the RETURN event on the `THIS-PROCEDURE` handle.

- If you are using ABL with a DataServer that supports stored procedures, the `RUN` statement has extensions that allow you to execute a stored procedure. For more information, see the entry for the `RUN STORED-PROCEDURE` statement and the appropriate OpenEdge DataServer Guides (*Use the Microsoft SQL Data Server* and *Use the Oracle Data Server*).

- If you `RUN` a procedure multiple times within a session, changing the procedure between runs, you must manually recompile the procedure each time. Otherwise, the procedure's last r-code, which persists for a session, is what is run and the changes do not appear.

- An asynchronous call to a remote procedure (using the `ASYNCHRONOUS` option) causes the `RUN` statement to return control immediately to the following statement in the local context, whether or not the remote procedure has completed execution.

- If an asynchronous call to a remote procedure does not raise a `STOP` or `ERROR` condition, the AVM:

  - Increments the `server-handle:ASYNC-REQUEST-COUNT` attribute
  - Increments the `proc-handle:ASYNC-REQUEST-COUNT` attribute, if `PERSISTENT` is specified for a remote external procedure or `IN proc-handle` is specified for a remote internal procedure
  - Sets the `async-request-handle:COMPLETE` attribute to FALSE, indicating that the request has not completed execution
  - Sets the `async-request-handle:EVENT-PROCEDURE` attribute to the value of `event-internal-procedure`, if `event-internal-procedure` is specified
  - Sets the `async-request-handle:EVENT-PROCEDURE-CONTEXT` attribute to the value of `procedure-context`, if `procedure-context` is specified
  - Submits the request for execution by the application server

- An asynchronous call to a remote procedure can return a class-based error object. The object is returned via the `ERROR-OBJECT` attribute on the Asynchronous request object handle.

- The AVM checks the syntax of the `ON SERVER` option at run time. This allows you to use a single HANDLE variable that you can set either to a server handle value or the value of the current `SESSION` handle. Thus, you can use the same `RUN` statement to execute a procedure remotely in an application server or locally depending on application conditions.

- When you specify the ON SERVER option with the SESSION system handle, the RUN statement is functionally similar to not specifying the ON SERVER option at all. That is, the two RUN statements in the following code perform the same function:

```
DEFINE VARIABLE hServer AS HANDLE NO-UNDO.

hServer = SESSION.
RUN foo.p.
RUN foo.p ON SERVER hServer.
```

  Allowing the same ON SERVER option to specify either a local session or a remote application server session facilitates code generation for applications like the OpenEdge AppBuilder.

  With the ASYNCHRONOUS option, using the ON SERVER SESSION option causes the called procedure to run **synchronously** in the local session, followed immediately by execution of any specified *event-handler-name*. Only after execution of the specified *event-handler-name* does control return to the statement following the RUN statement. This synchronous local execution includes the following differences in error handling from asynchronous execution on an application server using ON SERVER *server-handle*:

  - If an unhandled ERROR condition occurs during execution of the called local procedure or method, the error message is displayed on the local output device. This is different from remote execution, where any error message is written to the application server log file.
  - If the called local procedure or method causes an ERROR or STOP condition to be raised in the calling procedure (a file not found, mismatched parameters, a compile error, and explicit execution of a RETURN ERROR or STOP statement), the AVM sends the associated message to the standard output device and sets ERROR-STATUS:ERROR appropriately. This is different from remote execution, where the AVM in most cases attaches the associated message to the ERROR-STATUS system handle.
  - Also, if the called local procedure or method causes an ERROR or STOP condition to be raised in the calling procedure (as in the previous note), the AVM raises the condition on the RUN statement, as for a local RUN statement without the ON SERVER option. This is different from remote execution, where the AVM does not raise the condition on the calling RUN statement. You can work around this for the ON SERVER SESSION case by coding each asynchronous RUN statement with the NO-ERROR option and possibly surrounding it with a DO ON STOP UNDO, LEAVE block.

See also

{ } Argument reference, { } Include file reference, APPLY statement, Asynchronous request object handle, CODEBASE-LOCATOR system handle, COMPILE statement, CREATE SERVER statement, DEFINE PARAMETER statement, DELETE PROCEDURE statement, DELETE OBJECT statement, ERROR-OBJECT attribute, ON statement, Parameter passing syntax, PROCEDURE statement, Procedure object handle, RUN STORED-PROCEDURE statement, THIS-PROCEDURE system handle, VALID-HANDLE function, Widget phrase

RUN STORED-PROCEDURE statement

Runs a non-ABL stored procedure or allows you to send SQL to an SQL-based data source using an OpenEdge DataServer.

Syntax

```
RUN STORED-PROCEDURE procedure-name
     [ [ LOAD-RESULT-INTO handle [ status = PROC-STATUS ] ] |
     [ identifier = PROC-HANDLE ] ]
     [ NO-ERROR ]
     [ ( parameter [ , parameter ] . . .  ) ]
```

Note that not all options shown can be used simultaneously.

procedure-name

The name of the stored procedure that you want to run or the ABL built-in procedure name `send-sql-statement`.

Use `send-sql-statement` to send an SQL statement to an SQL-based data source.

`LOAD-RESULT-INTO` *handle*

Loads the results using a *handle* which points to one or more temp-tables. Note that the handle can be defined as an `EXTENT`. This enables you to pass more than one temp-table handle in those instances where SQL statement(s), or named stored procedures, return more than one result set. You can pass an unprepared temp-table (that is, no defined schema) with `LOAD-RESULT-INTO`. In such cases, the temp-table structure is prepared dynamically. If the temp-table defined by the handle already has a prepared schema, the temp-table structure must match the structure of the result set to avoid data mismatch errors. See the "Data types" section in the "Initial Programming Considerations" of your respective *DataServer Guide* for data mapping requirements. If multiple result sets are returned, each result set needs its own temp-table mapping. The elements of a temp-table extent should be arranged in the same order that the procedure returns its sets of results.

`LOAD-RESULT-INTO` carries an implicit `CLOSE STORED-PROCEDURE` statement.

If *handle* does not point to a temp-table, a run-time error occurs.

status `= PROC-STATUS`

Assigns an integer value to *status* containing the return status from the stored procedure. The status is typically a code indicating whether the procedure succeeded or failed and why.

identifier `= PROC-HANDLE`

Assigns a value to *identifier*, of the appropriate data type (usually INTEGER), that uniquely identifies the stored procedure returning results from the non-OpenEdge database or that uniquely identifies the SQL cursor used to retrieve results from an SQL-based data source, such as a DataServer for Microsoft SQL Server.

`NO-ERROR`

The NO-ERROR option is used to prevent the statement from raising `ERROR` and displaying error messages.

Note: This option must appear before any run-time parameter list.

`parameter`

A run-time parameter to be passed to the stored procedure. A *`parameter`* has the following syntax:

```
[ INPUT | OUTPUT | INPUT-OUTPUT ]
    [ PARAM parameter-name = ] expression ) ]
```

INPUT is the default. OUTPUT and INPUT-OUTPUT parameters must be record fields or program variables. For ORACLE, OUTPUT and INPUT-OUTPUT work the same way.

`parameter-name` is the name of a keyword parameter defined by the stored procedure. If not specified you must supply all the parameters in the correct order and you must precede your assignment statement with the PARAM keyword. If you do not supply a required parameter, and no default is specified in the stored procedure, a run-time error occurs.

`expression` is a constant, field name, variable name, or expression. If you use `send-sql-statement` for an SQL-based data source, you must pass only one character expression containing the SQL statement you want the data source to execute.

Examples

This procedure runs the ORACLE stored procedure `pcust` and writes the results of the stored procedure into the ABL-supplied buffer, `proc-text-buffer`. The same code works for accessing a stored procedure from a DataServer for Microsoft SQL Server data source.

```
DEFINE VARIABLE intvar AS INTEGER NO-UNDO.

RUN STORED-PROCEDURE pcust intvar = PROC-HANDLE NO-ERROR
  (10, OUTPUT 0, OUTPUT 0).
FOR EACH proc-text-buffer WHERE PROC-HANDLE = intvar:
  DISPLAY proc-text-buffer.
END.
IF ERROR-STATUS:ERROR THEN
  MESSAGE "Stored Procedure failed to run".
ELSE
  CLOSE STORED-PROCEDURE pcust WHERE PROC-HANDLE = intvar.
```

This procedure uses the `send-sql-statement` option to send SQL to ORACLE. It writes the results of the stored procedure into the ABL-supplied buffer, `proc-text-buffer`. The same code works for sending SQL to a DataServer for Microsoft SQL Server data source:

```
DEFINE VAR handle1 AS INTEGER.

RUN STORED-PROCEDURE send-sql-statement handle1 = PROC-HANDLE
  ("SELECT name, cust_num FROM customer").
FOR EACH proc-text-buffer WHERE PROC-HANDLE = handle1:
  DISPLAY proc-text.
END.

CLOSE STORED-PROCEDURE send-sql-statement WHERE PROC-HANDLE = handle1.
```

This example shows how to trap errors from the non-OpenEdge RDBMS within a procedure:

```
DEFINE VAR h1 AS INTEGER NO-UNDO.
DEFINE VAR jx AS INTEGER NO-UNDO.

RUN STORED-PROCEDURE send-sql-statement h1 = PROC-HANDLE NO-ERROR
  ("select count (*) from xxx.customer where name between 'A' and 'Z' ").
IF ERROR-STATUS:ERROR THEN
DO jx = 1 TO ERROR-STATUS:NUM-MESSAGES:
  MESSAGE "error" ERROR-STATUS:GET-NUMBER(jx)
    ERROR-STATUS:GET-MESSAGE(jx).
END.

CLOSE STORED-PROCEDURE send-sql-statement WHERE PROC-HANDLE = h1.
```

This example shows how to use the `send-sql-statement` with the `LOAD-RESULT-INTO` option. Note that the temp-table created for `tthndl` is unprepared. When `tthndl` is given to `LOAD-RESULT-INTO` and the temp-table is unprepared, `LOAD-RESULT-INTO` prepares the temp-table when results are returned. Note also that you do not have to specify a `CLOSE STORED-PROCEDURE` statement, since it's done implicitly with `LOAD-RESULT-INTO`.

```
DEFINE VARIABLE res    AS INTEGER NO-UNDO INITIAL 0.
DEFINE VARIABLE tthndl AS HANDLE.

CREATE TEMP-TABLE tthndl.

RUN STORED-PROCEDURE send-sql-statement LOAD-RESULT-INTO tthndl
  res = PROC-STATUS ("SELECT * FROM customer").

IF (res = 1) THEN
  DISPLAY "Succeeded".
```

Notes

- The `RUN STORED-PROCEDURE` statement starts a transaction with the same scope as transactions started with the `UPDATE` statement.
- For more information on using this statement and on using the built-in procedure names, `send-sql-statement`, see the OpenEdge DataServer Guides (*Use the Microsoft SQL Data Server* and *Use the Oracle Data Server*).

See also

CLOSE STORED-PROCEDURE statement, NO-ERROR option, PROC-HANDLE function, PROC-STATUS function

RUN SUPER statement

Runs the super procedure version of the current internal procedure.

The `RUN SUPER` statement must appear only within an internal procedure, but can appear anywhere within the internal procedure. If the `RUN SUPER` statement appears outside an internal procedure, the compiler reports an error.

Syntax

```
RUN SUPER [ ( parameter[ , parameter]... ) ][ NO-ERROR ]
```

parameter

A parameter of the super procedure. The parameters of the super procedure must have the same signature (number of parameters, and type and mode of each) as the parameters of the current internal procedure. You can, however, adjust a parameter's value.

For the *parameter* syntax, see the Parameter passing syntax reference entry.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

The following example consists of three procedure files: a main routine, a driver, and a third procedure file that becomes a super procedure of the driver.

The following main routine, procedure file `r-pomain.p`, runs the driver procedure persistently:

r-pomain.p

```
DEFINE VARIABLE h AS HANDLE    NO-UNDO.
DEFINE VARIABLE a AS CHARACTER NO-UNDO.

FUNCTION sample2 RETURNS CHARACTER (INPUT-OUTPUT a AS CHARACTER) IN h.

RUN r-podrvr.p PERSISTENT SET h.
RUN sample1 IN h (INPUT-OUTPUT a).

MESSAGE a VIEW-AS ALERT-BOX.
a = "".
MESSAGE sample2(a) VIEW-AS ALERT-BOX.
```

The following driver, procedure file `r-podrvr.p`, runs the third procedure file persistently, makes it a super procedure of itself, defines the internal procedure sample1, and defines the user-defined functions sample2, GetPartName, and SetPartName:

r-podrvr.p

```
DEFINE VARIABLE h AS HANDLE.DEFINE VARIABLE localPartName AS CHARACTER.

FUNCTION SetPartName RETURNS INTEGER (INPUT a AS CHARACTER) FORWARD.

/* Add a super procedure */
RUN r-posupr.p PERSISTENT SET h.
THIS-PROCEDURE:ADD-SUPER-PROCEDURE (h).
SetPartName("1998 Calendar").

PROCEDURE sample1:
  DEFINE INPUT-OUTPUT PARAMETER a AS CHARACTER NO-UNDO.

  a = a + "proc: Part name is: ".
  /* Invoke procedure sample1 in the super procedure. */
  RUN SUPER (INPUT-OUTPUT a).
END PROCEDURE.

FUNCTION sample2 RETURNS CHARACTER (INPUT-OUTPUT a AS CHARACTER).
  a = a + "func: Part name is: ".
  /* Invoke function sample2 in the super procedure. */
  SUPER (INPUT-OUTPUT a).
  RETURN a.
END FUNCTION.

FUNCTION GetPartName RETURNS CHARACTER ():
  RETURN localPartName.
END FUNCTION.

FUNCTION SetPartName RETURNS INTEGER (INPUT partname AS CHARACTER):
  localPartName = partname.
END FUNCTION.
```

The following third procedure file, `r-posupr.p`, defines a new version of the internal procedure sample1 and a new version of the user-defined function sample2:

r-posupr.p

```
/* r-posupr.p */
DEFINE VARIABLE h AS HANDLE NO-UNDO.

FUNCTION GetPartName RETURNS CHARACTER () IN H.

PROCEDURE sample1:
  DEFINE INPUT-OUTPUT PARAMETER a AS CHARACTER NO-UNDO.

  h = TARGET-PROCEDURE.
  a = a + GetPartName().
  MESSAGE "TARGET-PROCEDURE is:" TARGET-PROCEDURE:FILE-NAME
    VIEW-AS ALERT-BOX.
  MESSAGE "SOURCE-PROCEDURE is:" SOURCE-PROCEDURE:FILE-NAME
    VIEW-AS ALERT-BOX.
END PROCEDURE.

FUNCTION sample2 RETURNS CHARACTER (INPUT-OUTPUT a AS CHARACTER):
  h = TARGET-PROCEDURE.
  a = a + GetPartName().
  RETURN a.
END.
```

To start the example, run `r-pomain.p` from the Procedure Editor.

Notes

- To run the super version of a user-defined function, use the SUPER function.
- For the rules that ABL uses to find the super procedure, see the reference entry for the ADD-SUPER-PROCEDURE( ) method.

See also

ADD-SUPER-PROCEDURE( ) method, NO-ERROR option, REMOVE-SUPER-PROCEDURE( ) method, SOURCE-PROCEDURE system handle, SUPER function, SUPER-PROCEDURES attribute, TARGET-PROCEDURE system handle

SAVE CACHE statement

Saves the schema cache of a database to an operating system file. Subsequent sessions can then share the same cache by using the Schema Cache File (`-cache`) parameter.

Syntax

```
SAVE CACHE
    { CURRENT | COMPLETE }
    {database-name | VALUE ( char-expr ) }
  TO
    {pathname |  VALUE ( char-expr ) }
[ NO-ERROR ]
```

`CURRENT`

Specifies that only the portion of the schema cache that applies to referenced tables is saved to the file. By using this option you can tailor a small schema cache file for an application that does not use all the tables in the database.

`COMPLETE`

Specifies that the complete schema cache for the database is saved to the file. If you use this option, the client process builds a complete schema cache in memory including template records and all trigger information for every table in the database.

`database-name`

Specifies the literal logical name of a currently connected OpenEdge database.

`pathname`

Specifies the literal pathname of an operating system file to hold the schema cache. In Windows, the pathname cannot contain characters outside of the non-Unicode code page. See *Internationalize ABL Applications* for more information about Unicode and code pages.

VALUE (*char-expr*)

Returns the corresponding literal database name or pathname specified by the character expression in *char-expr*.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

This procedure saves the complete schema cache for each database that you specify in the current working directory, and displays any error messages associated with connecting or saving the cache:

r-schcsh.p

```
DEFINE VARIABLE db-name AS CHARACTER NO-UNDO FORMAT "x(12)" INITIAL ?.
DEFINE VARIABLE icnt    AS INTEGER   NO-UNDO.

DO WHILE db-name <> "":
 SET db-name LABEL "Database Name"
   WITH FRAME A SIDE-LABELS TITLE "Save Cache" VIEW-AS DIALOG-BOX.
 IF db-name <> "" THEN
   CONNECT VALUE(db-name) -1 NO-ERROR.
 ELSE LEAVE.

 IF NOT ERROR-STATUS:ERROR THEN DO:
   SAVE CACHE COMPLETE VALUE(db-name) TO VALUE(db-name + ".csh") NO-ERROR.
   IF NOT ERROR-STATUS:ERROR THEN
     MESSAGE "Saved schema cache for" db-name "in" db-name + ".csh.".
   ELSE DO:
     BELL.
     DO icnt = 1 TO ERROR-STATUS:NUM-MESSAGES:
       MESSAGE ERROR-STATUS:GET-MESSAGE(icnt) VIEW-AS ALERT-BOX.
     END.
   END.
 END.
 ELSE DO:
   BELL.
   DO icnt = 1 TO ERROR-STATUS:NUM-MESSAGES:
     MESSAGE ERROR-STATUS:GET-MESSAGE(icnt) VIEW-AS ALERT-BOX.
   END.
 END.
 DISCONNECT VALUE(db-name) NO-ERROR.
END.
```

Notes

- The schema cache is saved to the file in a binary format that is portable across machines.
- For information on using an existing schema cache file, see *Manage the OpenEdge Database*. For information on the Schema Cache File (-cache) startup parameter, see *Startup Command and Parameter Reference*.
- Any schema changes to the database make the saved cache invalid. If the schema cache file is invalid when the AVM tries to access it, the AVM displays a warning message, ignores the file, and reads the required schema cache from the database.
- To set up your database environment to use the CURRENT option, you only have to connect to the database and read from the tables that compose the schema you want to save. This is sufficient for the SAVE CACHE statement to save all parts of each table in the schema, including template records and trigger information.

If you want to save a different subschema of the database, you must disconnect and then reconnect to the database before reading the tables for that subschema.

- For a DataServer, the AVM saves the schema cache for the entire schema holder database. You cannot save the schema cache for a non-OpenEdge database separately. For more information on schema cache files for DataServers, see the OpenEdge DataServer Guides (*Use the Microsoft SQL Data Server* and *Use the Oracle Data Server*).

See also

CONNECT statement, ERROR-STATUS system handle, NO-ERROR option

SCREEN-LINES function

Returns, as an INTEGER value, the number of lines you can use to display frames. This value omits the space used by the message area and status area.

Note: Does not apply to SpeedScript programming.

Syntax

```
SCREEN-LINES
```

Example

Here, a different number of customer records is displayed depending on the number returned by the SCREEN-LINES function:

r-scrnln.p

```
DEFINE VARIABLE nbrdown AS INTEGER NO-UNDO.

nbrdown = IF SCREEN-LINES > 21 THEN 7 ELSE 6.

FOR EACH Customer NO-LOCK WITH nbrdown DOWN:
  DISPLAY Customer.CustNum Customer.Name Customer.Address Customer.City
    Customer.State Customer.Country.
END.
```

SCROLL statement

Moves data up or down in a frame with multiple rows. Use the SCROLL statement to scroll data up or down when you add or delete a line in a frame.

This statement is supported only for backward compatibility.

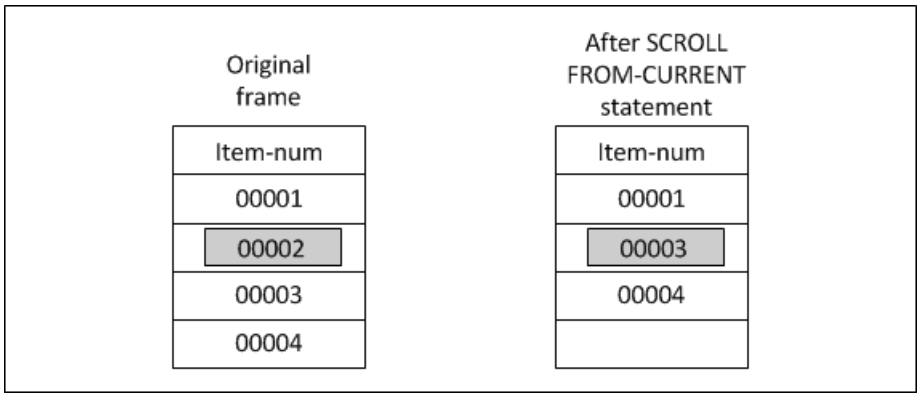
Note: Does not apply to SpeedScript programming.

Syntax

```
SCROLL [ FROM-CURRENT ][ UP | DOWN ]{[frame-phrase]}
```

FROM-CURRENT

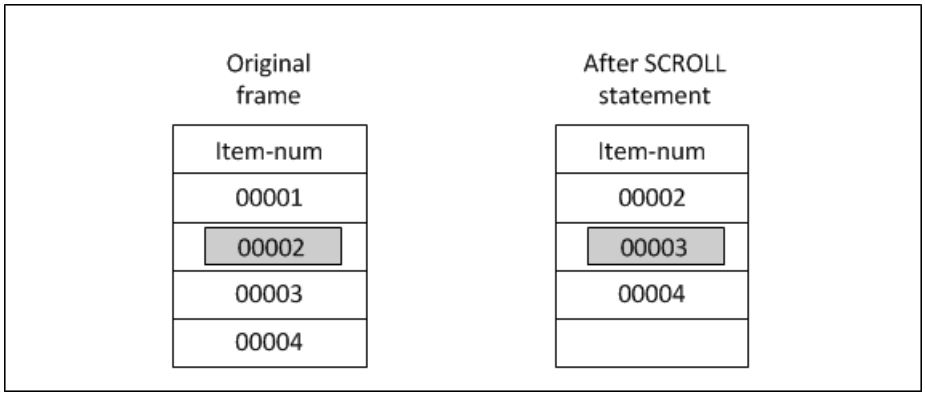
Scrolls UP or DOWN rows of data at or below the current cursor location. When scrolling UP, a new line opens at the bottom of the frame. When scrolling DOWN, a new line opens at the current cursor location. For example:



If you do not use the FROM-CURRENT option, then the entire frame scrolls up or down and the newly opened line appears at the top or bottom of a frame, respectively. FROM-CURRENT limits scrolling from the current cursor position to the bottom of the frame.

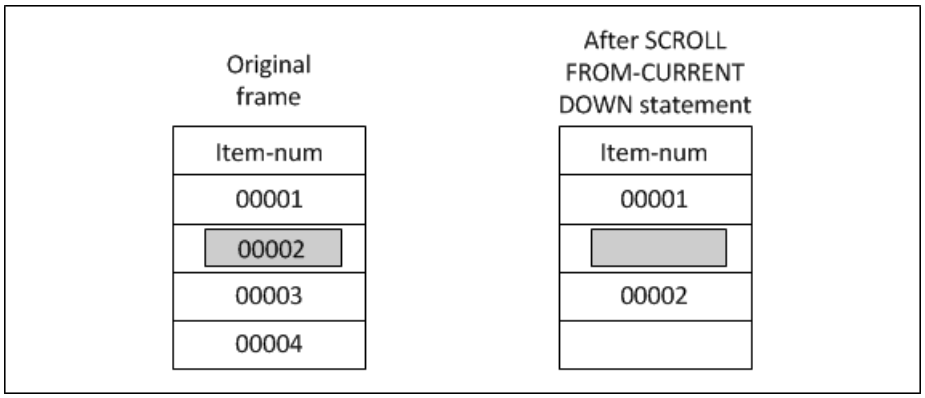
UP

Scrolls rows of data up and off the frame and opens a line at the bottom of the frame. UP is the default. For example:

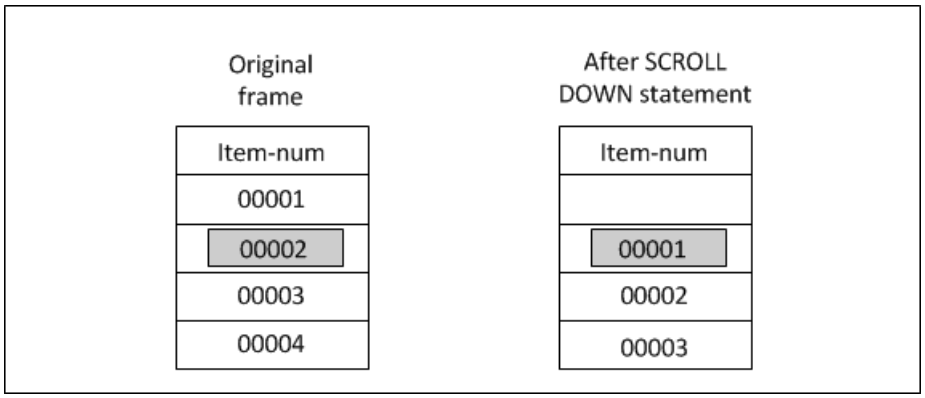


DOWN

Scrolls rows of data down and off the frame and opens a line at the top of the frame. For example, the Original Frame in the next example shows four rows of data. The highlighted bar is the current cursor position and the frame is a scrolling frame. On the right, the SCROLL FROM-CURRENT DOWN statement opens a line in the frame at the current cursor location and moves the other rows down and off the frame. For example:



In the next example, the SCROLL DOWN statement opens a line at the top of the frame and moves the other rows of data down and off the frame:



frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

Examples

This procedure displays Customer information and lets you try each scrolling option from a menu of selections:

r-scroll.p

```
DEFINE VARIABLE ans AS CHARACTER NO-UNDO FORMAT "x".

FORM Customer.CustNum Customer.Name Customer.CreditLimit
  WITH FRAME cust CENTERED 10 DOWN.

FORM
  "1 - scroll up" SKIP
  "2 - scroll from-current up" SKIP
  "3 - scroll down" SKIP
  "4 - scroll from-current down" SKIP
  "5 - scroll from-current "
  WITH FRAME instruct CENTERED TITLE "Instructions:".
VIEW FRAME cust.

REPEAT WHILE FRAME-LINE(cust) <= FRAME-DOWN(cust):
  FIND NEXT Customer NO-LOCK.
  DISPLAY Customer.CustNum Customer.Name Customer.CreditLimit
    WITH FRAME cust TITLE "Customers".
  DOWN WITH FRAME cust.
END.

UP FRAME-DOWN(cust) / 2 WITH FRAME cust.

VIEW FRAME instruct.REPEAT WITH FRAME cust:
  CHOOSE ROW Customer.Name KEYS ans AUTO-RETURN NO-ERROR WITH FRAME cust.
       IF ans = "1" THEN SCROLL UP.
  ELSE IF ans = "2" THEN SCROLL FROM-CURRENT UP.
  ELSE IF ans = "3" THEN SCROLL DOWN.
  ELSE IF ans = "4" THEN SCROLL FROM-CURRENT DOWN.
  ELSE IF ans = "5" THEN SCROLL FROM-CURRENT.
  VIEW FRAME cust.
  ans = "".
END.
```

The next procedure creates a scrolling frame of five iterations. The frame displays the `CustNum`, `Name`, `Address`, and `City` for each Customer. The status default message displays "Enter C to create, D to delete" as long as the procedure is running. You use arrow keys to move the highlighted cursor bar through the database, and to add or delete Customers from the database. The CHOOSE statement lets you easily create this style menu. See the CHOOSE statement reference entry for more information.

r-chose1.p

```
DEFINE VARIABLE counter   AS INTEGER   NO-UNDO.
DEFINE VARIABLE oldchoice AS CHARACTER NO-UNDO.

FORM Customer.CustNum Customer.Name Customer.Address Customer.City
  WITH FRAME cust-frame SCROLL 1 5 DOWN.

FIND FIRST Customer NO-LOCK.

REPEAT counter = 1 TO 5:
  DISPLAY Customer.CustNum Customer.Name Customer.Address Customer.City
    WITH FRAME cust-frame.
  DOWN WITH FRAME cust-frame.
  FIND NEXT Customer NO-LOCK NO-ERROR.
  IF NOT AVAILABLE customer THEN LEAVE.
END.
UP 5 WITH FRAME cust-frame.
oldchoice = "".

REPEAT:
  STATUS DEFAULT "Enter C to create, D to delete".
```

```
CHOOSE ROW Customer.CustNum NO-ERROR GO-ON(CURSOR-RIGHT)
  WITH FRAME cust-frame.

/* After choice */
IF FRAME-VALUE = "" THEN NEXT.

/* Force user to press END or move cursor to valid line */
IF FRAME-VALUE <> oldchoice THEN DO:
  oldchoice = FRAME-VALUE.
  FIND Customer WHERE Customer.CustNum = INTEGER(FRAME-VALUE).
END.

/* React to moving cursor off the screen */
IF LASTKEY = KEYCODE("CURSOR-DOWN") THEN DO:
  FIND NEXT customer NO-ERROR.
  IF NOT AVAILABLE Customer THEN
    FIND FIRST Customer NO-LOCK.
  DOWN WITH FRAME cust-frame.
  DISPLAY Customer.CustNum Customer.Name Customer.Address Customer.City
    WITH FRAME cust-frame.
  NEXT.
END. /* KEYCODE("CURSOR-DOWN") */

IF LASTKEY = KEYCODE("CURSOR-UP") THEN DO:
  FIND PREV Customer NO-LOCK NO-ERROR.
  IF NOT AVAILABLE Customer THEN
    FIND LAST Customer NO-LOCK.
  UP WITH FRAME cust-frame.
  DISPLAY Customer.CustNum Customer.Name Customer.Address Customer.City
    WITH FRAME cust-frame.
  NEXT.
END. /* KEYCODE("CURSOR-UP") */

/* CHOOSE selected a valid key. Check which key. */
IF LASTKEY = KEYCODE("c") THEN DO: /* Open a space in the frame. */
  SCROLL FROM-CURRENT DOWN WITH FRAME cust-frame.
  CREATE Customer.
  UPDATE Customer.CustNum Customer.Name Customer.Address Customer.city
    WITH FRAME cust-frame.
  oldchoice = INPUT Customer.CustNum.
  NEXT.
END. /* KEYCODE("c") */

IF LASTKEY = KEYCODE("d") THEN DO: /* Delete a customer. */
  DELETE Customer.
  FIND NEXT Customer NO-ERROR.
  /* Move to correct position in database. */
  IF NOT AVAILABLE Customer THEN DO:
    FIND FIRST Customer NO-ERROR.
    IF NOT AVAILABLE Customer THEN DO:
      CLEAR FRAME cust-frame.
      UP WITH FRAME cust-frame.
      NEXT.
    END.
  END.

  /* If last screen line deleted */
  IF FRAME-LINE(cust-frame) = FRAME-DOWN(cust-frame) THEN DO:/*
    DISPLAY Customer.CustNum Customer.Name Customer.Address Customer.City
      WITH FRAME cust-frame.
    NEXT.
  END.

  SCROLL FROM-CURRENT WITH FRAME cust-frame.
  REPEAT counter = 1 TO 100
    WHILE FRAME-LINE(cust-frame) < FRAME-DOWN(cust-frame):
    FIND NEXT Customer NO-ERROR.
    IF NOT AVAILABLE Customer THEN DO:
      FIND FIRST Customer NO-ERROR.
```

```
        IF NOT AVAILABLE Customer THEN LEAVE.
      END.
      DOWN WITH FRAME cust-frame.
      IF INPUT Customer.CustNum = "" THEN
        DISPLAY Customer.CustNum Customer.Name Customer.Address
          Customer.City WITH FRAME cust-frame.
    END.
    UP counter - 1 WITH FRAME cust-frame.
    oldchoice = INPUT Customer.CustNum.
  END. /* KEYCODE("d") */
END. /* REPEAT */

STATUS DEFAULT.
```

The SCROLL statement controls the scrolling action in the frame when you create and delete customers. To add a customer to the database, type `C`. Create opens a line in the frame and the SCROLL statement moves data below the line down. Then you type the new customer information into the frame. Type `D` to delete a Customer from the database. When you delete a Customer, all rows below the deleted Customer row move up one row.

You can perform the same function with fewer statements if you do not use the SCROLL statement. You can substitute the `r-chose1.p` procedure segment with the `r-chose2.p` to perform the delete function.

r-chose2.p

```
. . .
IF LASTKEY = KEYCODE("d") THEN DO: /* Delete a customer. */
  DELETE Customer.
  REPEAT counter = 1 TO 100
    WHILE FRAME-LINE(cust-frame) <= FRAME-DOWN(cust-frame).
    FIND NEXT Customer NO-ERROR.
    IF AVAILABLE Customer THEN
      DISPLAY Customer.CustNum Customer.Name Customer.Address Customer.City
       WITH FRAME cust-frame.
    ELSE CLEAR FRAME cust-frame.
    DOWN WITH FRAME cust-frame.
  END.
  UP counter - 1 WITH FRAME cust-frame.
  oldchoice = INPUT Customer.CustNum.
END. /* KEYCODE("d") */
. . .
```

You can see the entire `r-chose2.p` procedure on-line. This example only shows the portion that is different from the `r-chose1.p` procedure.

The `r-cuhelp.p` procedure provides help for the `CustNum` field when a user presses **HELP**. It displays five Customer names and numbers. The user can press (**UP-ARROW**), (**DOWN-ARROW**), to scroll down, or (**RETURN**) to exit.

r-cuhelp.p

```
FORM Customer.CustNum Customer.Name
  WITH FRAME cust-frame 5 DOWN ROW 10 CENTERED
  OVERLAY TITLE " Available Customers ".

REPEAT WHILE FRAME-LINE(cust-frame) <= FRAME-DOWN(cust-frame):
  FIND NEXT Customer.
  DISPLAY Customer.CustNum Customer.Name WITH FRAME cust-frame.
  DOWN WITH FRAME cust-frame.
END.

UP 5 WITH FRAME cust-frame.

REPEAT:
  CHOOSE ROW Customer.CustNum NO-ERROR WITH FRAME cust-frame.
  FIND Customer WHERE Customer.CustNum = INPUT Customer.CustNum.
  IF KEYFUNCTION(LASTKEY) = "CURSOR-UP" THEN DO:
    FIND PREV Customer NO-ERROR.
    IF AVAILABLE Customer THEN DO:
      SCROLL DOWN WITH FRAME cust-frame.
      DISPLAY Customer.CustNum Customer.Name WITH FRAME cust-frame.
    END.
  END.
  ELSE
  IF KEYFUNCTION(LASTKEY) = "CURSOR-DOWN" THEN DO:
    FIND NEXT Customer NO-ERROR.
    IF AVAILABLE Customer THEN DO:
      SCROLL UP WITH FRAME cust-frame.
      DISPLAY Customer.CustNum Customer.Name
        WITH FRAME cust-frame.
    END.
  END.
  ELSE
  IF KEYFUNCTION(LASTKEY) = "RETURN" THEN DO:
    FRAME-VALUE = FRAME-VALUE.
    HIDE FRAME cust-frame.
    RETURN.
  END.
END.
```

See also

CHOOSE statement, Frame phrase

SDBNAME function

Accepts an integer expression or a character expression as a parameter. If the parameter resolves to a currently connected non-OpenEdge database then the SDBNAME function returns the logical name of the schema holder database containing the non-OpenEdge schema. If the parameter resolves to a currently connected OpenEdge database, the SDBNAME function returns the logical name of this database.

Syntax

```
SDBNAME ( { integer-expression | logical-name | alias } )
```

integer-expression

If the parameter supplied to SDBNAME is an integer expression, and there are, for example, three connected databases, then SDBNAME(1), SDBNAME(2), and SDBNAME(3) return the logical names of their respective schema holder databases. Also, if there are three connected databases, SDBNAME(4), SDBNAME(5), etc., return the Unknown value (?).

logical-name or *alias*

These forms of the SDBNAME function require a quoted character string or a character expression as a parameter. If the parameter is the logical name of a connected database or an alias of a connected database, then the logical name of the schema holder database is returned according to the rule. Otherwise, SDBNAME returns the Unknown value (?).

Example

This procedure displays schema holder databases, if applicable, for all connected databases:

r-sdbnm.p

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

REPEAT ix = 1 TO NUM-DBS:
  DISPLAY SDBNAME(ix) SDBNAME(ix) = LDBNAME(ix)
  FORMAT "SCHEMA-HOLDER/SUB-SCHEMA        "
    COLUMN-LABEL "  DataServer!Classification".
END.
```

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function

SEARCH function

Searches the directories and libraries defined in the PROPATH environment variable for a file. The SEARCH function returns the full pathname of the file unless it is found in your current working directory. If SEARCH does not find the file, it returns the Unknown value (?).

Syntax

```
SEARCH ( opsys-file )
```

opsys-file

A character expression whose value is the name of the file you want to find. The name can include a complete or partial directory path. If *opsys-file* is a constant string, you must enclose it in quotation marks (" "). The value of *opsys-file* must be no more than 255 characters long. The filename can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

Example

In this procedure, the SEARCH function returns the fully qualified pathname of the filename entered if it is not in the current working directory. If SEARCH cannot find the file, it returns the Unknown value (?). The procedure displays the fully qualified pathname or a message indicating that the file could not be found.

r-search.p

```
DEFINE VARIABLE fullname AS CHARACTER NO-UNDO FORMAT "x(55)".
DEFINE VARIABLE filename AS CHARACTER NO-UNDO FORMAT "x(20)".

REPEAT:
  UPDATE filename HELP "Try entering 'help.r' or 'dict.r'"
    WITH FRAME a SIDE-LABELS CENTERED.
    fullname = SEARCH(filename).
    IF fullname = ? THEN
      DISPLAY "UNABLE TO FIND FILE " filename
        WITH FRAME b ROW 6 CENTERED NO-LABELS.
    ELSE
      DISPLAY "Fully Qualified Path Name Of:" filename SKIP(2)
        "is:" fullname WITH FRAME c ROW 6 NO-LABELS CENTERED.
END.
```

Notes

- The SEARCH function is double-byte enabled. You can specify a filename with the *opsys-file* argument that contains double-byte characters.
- Use the SEARCH function to ensure that procedures that get input from external data files are independent of specific directory paths. The files must be in one of the directories or libraries defined in the PROPATH environment variable.
- Typically, the PROPATH includes a nil entry representing the current working directory. If the SEARCH function finds the file when searching this entry, it returns only the simple name of the file rather than the full pathname. If the PROPATH does not include a nil entry or another entry that specifies the current working directory, the SEARCH function does not search the current working directory.
- If you provide a fully qualified pathname, SEARCH checks if the file exists. In this case, SEARCH does not search directories on the PROPATH.
- When you search for a file that is in a library, SEARCH returns the file's pathname in the form *path-name*<<*member-name*>>, where *path-name* is the pathname of the library and *member-name* is the name of the file. The double angle brackets indicate that the file is a member of a library. For example, in the path `/usr/apps.pl<<proc1.r>>`, proc1.r is the name of the file in the library apps.pl.

  The LIBRARY function and MEMBER function use the special syntax to return, respectively, the library name and *member-name* of the file in the library.
- If an application repeatedly runs a procedure, you can improve performance by using the SEARCH function once to build a full pathname for that procedure. Use this value in the RUN statement to avoid repeated searches of the PROPATH.
- In Windows, you can specify URL pathnames on the PROPATH. If the file is found in a directory specified by a URL, SEARCH returns the full URL pathname of the file which includes the filename appended to the URL PROPATH entry. If you provide a fully-qualified URL, SEARCH checks if the file exists. In this case, SEARCH does not search URLs on the PROPATH. Valid URL protocols include HTTP and HTTPS.

  Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, the SEARCH function continues searching with the next PROPATH entry.

- If you specify URL pathnames on the PROPATH and your application repeatedly uses the LOAD-ICON( ), LOAD-SMALL-ICON( ), LOAD-IMAGE( ), LOAD-IMAGE-DOWN( ), LOAD-IMAGE-UP( ), LOAD-IMAGE-INSENSITIVE( ), or LOAD-MOUSE-POINTER( ) methods with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the image files. Use this value with the load methods to avoid repeated searches of the PROPATH.

SEEK function

Returns the offset of the file pointer in a text file as an INT64 value. You define a procedure variable to hold the offset value and later position the file to that offset.

Syntax

```
SEEK ( { INPUT | OUTPUT | name | STREAM-HANDLE handle } )
```

INPUT

If you specify INPUT, the SEEK function returns the current position of the file pointer in the unnamed input stream.

OUTPUT

If you specify OUTPUT, the SEEK function returns the current position of the file pointer in the unnamed output stream.

name

If you specify SEEK (*name*), the SEEK function returns the current position of the file pointer in the named input or output stream. The stream must be associated with an open file, or SEEK returns the Unknown value (?).

STREAM-HANDLE *handle*

If you specify the handle to a stream, the SEEK function returns the current position of the file pointer in the stream. If *handle* it is not a valid handle to a stream, the AVM generates a run-time error.

Example

This procedure shows how you can use the SEEK function to access data in an text file. Using SEEK this way allows you to index into a non-indexed file.

r-seek1.p

```
DEFINE VARIABLE itemno NO-UNDO LIKE Item.ItemNum.
DEFINE VARIABLE itdesc NO-UNDO LIKE Item.ItemName.
DEFINE VARIABLE m-pos  AS INT64 NO-UNDO.

SET itemno LABEL
  "Select a record number to position the output file" WITH SIDE-LABELS.
OUTPUT TO test.fil.
FIND Item NO-LOCK WHERE Item.ItemNum = itemno.
IF Item.ItemNum = itemno THEN
  m-pos = SEEK(OUTPUT).
EXPORT Item.ItemNum Item.ItemName.
OUTPUT CLOSE.

INPUT FROM test.fil.
SEEK INPUT TO m-pos.
SET itemno itdesc WITH FRAME d2.
INPUT CLOSE.
```

In the example, you are prompted to select an Item number to position the output file. When a record is found with that Item number, the SEEK function returns the offset into the variable `m-pos`. The value for `m-pos` is the current value of the file pointer. The SEEK statement uses the value in `m-pos` to position the file pointer in the unnamed input stream.

Notes

- The first byte in a file is byte 0.
- You cannot use the SEEK function with the INPUT THROUGH statement, the INPUT-OUTPUT THROUGH statement, or the OUTPUT THROUGH statement. When used with one of these statements, the SEEK function returns the Unknown value (?).
- After you assign the value of the SEEK function to a procedure variable, you can use that value to reposition the file in the event of an error.
- For more information on streams, see the topics on alternate I/O sources in *OpenEdge Programming Interfaces*.

See also

DEFINE STREAM statement, INPUT FROM statement, OUTPUT TO statement, SEEK statement, Stream object handle

SEEK statement

Positions the file pointer to a user-defined offset in a text file. This statement does not require you to close and reopen the file.

Syntax

```
SEEK
  { INPUT | OUTPUT | STREAM stream | STREAM-HANDLE handle }
  TO {expression | END }
```

INPUT

If you specify INPUT, the SEEK statement positions the file pointer in the unnamed input stream.

OUTPUT

If you specify OUTPUT, the SEEK statement positions the file pointer in the unnamed output stream.

STREAM *stream*

If you specify STREAM *stream*, the SEEK statement positions the file pointer in the named input or output stream. If you do not name a stream, the AVM uses the unnamed stream.

STREAM-HANDLE *handle*

If you specify the handle to a stream, the SEEK statement returns the current position of the file pointer in the stream. If *handle* it is not a valid handle to a stream, the AVM generates a run-time error.

TO *expression*

An expression whose value is INT64 that indicates the byte location to position the file pointer. If *expression* equals 0, the file pointer is positioned to the first byte in the file. If you want to position the pointer to the last byte in the file, but you do not know the offset, use END.

END

Positions the pointer to the last byte in the file.

Example

Since text file formats differ on each machine, the SEEK function does not necessarily return a number that is meaningful to anyone, but it is meaningful to the SEEK statement. With the exception of SEEK to 0 or SEEK TO END, any address used in the SEEK statement is only guaranteed to behave consistently if the address was previously derived from the SEEK function. Therefore, an expression such as SEEK TO SEEK (INPUT) -n might work differently on different operating systems. Record delimiters must be new-lines on UNIX, and carriage-return/linefeed pairs on all others.

r-seek.p

```
/* This procedure seeks to the end-of-file, collects the seek address, and
   writes a record. The record is subsequently retrieved using the SEEK
   statement on the stashed seek address. */

DEFINE VARIABLE savepos AS INT64     NO-UNDO.
DEFINE VARIABLE c       AS CHARACTER NO-UNDO FORMAT "x(20)".

OUTPUT TO seek.out APPEND NO-ECHO.
savepos = SEEK(OUTPUT).
PUT UNFORMATTED "abcdefg" SKIP.
OUTPUT CLOSE.

INPUT FROM seek.out NO-ECHO.
SEEK INPUT TO savepos.
SET c.
DISPLAY c.
INPUT CLOSE.
```

Notes

- The SEEK statement does not work with named streams identified in the INPUT-THROUGH, OUTPUT-THROUGH, or INPUT-OUTPUT-THROUGH statements.
- An expression such as SEEK TO SEEK (INPUT) -n might work differently on different operating systems.
- For more information on streams, see the topics on alternate I/O sources in *OpenEdge Programming Interfaces*.

See also

DEFINE STREAM statement, INPUT FROM statement, OUTPUT TO statement, SEEK function, Stream object handle

SELECTION-LIST phrase

Describes the selection-list representation of a field or variable. A selection-list is a scrollable list of values. The SELECTION-LIST phrase is an option of the VIEW-AS phrase.

Note: Does not apply to SpeedScript programming.

Syntax

```
SELECTION-LIST
   [ SINGLE | MULTIPLE ]
   [ NO-DRAG ]
   { LIST-ITEMS item-list | LIST-ITEM-PAIRS item-pair-list }
   [ SCROLLBAR-HORIZONTAL ] [ SCROLLBAR-VERTICAL ]
   { size-phrase
     | { INNER-CHARS cols INNER-LINES rows }
   }
   [ SORT ]
   [ TOOLTIP tooltip ]
```

`SINGLE`

Specifies that on input the user can select only a single item from the list. This is the default. The value of the selection-list is set to the character-string item the user selects.

`MULTIPLE`

Specifies that on input the user can select one or more items from the item list. The value of the selection-list is set to a comma-separated list of character-string items that the user selects.

`NO-DRAG`

Specifies that the user cannot select items by simultaneously holding down the mouse select button and dragging the mouse through the list.

If you specify NO-DRAG then the DRAG-ENABLED attribute is set to FALSE. You can set the DRAG-ENABLED attribute only before the selection-list is realized. The default is TRUE.

In Windows, DRAG-ENABLED is always TRUE and the NO-DRAG option is ignored.

`LIST-ITEMS` *`item-list`*

Specifies the items to appear in the list. *`item-list`* represents a comma-separated list of character-string constants.

`LIST-ITEM-PAIRS` *`item-pair-list`*

Specifies a list of label-value pairs. Each pair represents a label and value of the associated field or variable. When the selection-list appears, it displays each pair's label. Then, if the user selects a label, the AVM assigns the corresponding value to the field or variable. The syntax for *`item-pair-list`* is as follows:

```
label , value [ , label , value ] ...
```

`label`

A character string representing the label of the field or variable.

`value`

A valid value for the field or variable.

`SCROLLBAR-VERTICAL`

Specifies that a scroll bar is displayed along side the selection-list. The user can browse through a long selection-list by manipulating the slider.

`SCROLLBAR-HORIZONTAL`

Specifies that a scroll bar is displayed along the bottom of the selection-list. The user can view long list items by manipulating the slider.

`size-phrase`

Specifies the outside dimensions of the selection-list widget. Following is the syntax for the *`size-phrase`*:

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

For more information, see the SIZE phrase reference entry.

`INNER-CHARS` *`cols`* `INNER-LINES` *`rows`*

Specifies the number of character positions visible in each line of the selection-list and the number of lines visible in the selection-list. Both *`cols`* and *`rows`* must be integer constants.

Note that the values you supply for INNER-CHARS and INNER-LINES specify only the size of the list, not the overall size of the selection-list widget. The overall size is determined by the size of the list plus the sizes of the margin and border heights and widths.

SORT

Specifies that list items are sorted prior to display.

TOOLTIP *tooltip*

Allows you to define a help text message for a text field or text variable. The AVM automatically displays this text when the user pauses the mouse button over a text field or text variable for which a tooltip is defined.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the tooltip is removed. No tooltip is the default. The TOOLTIP option is supported in Windows only.

Example

The `r-select.p` procedure prompts the user for a directory name and then populates a selection-list with the contents of the specified directory. After the user selects an item from the selection-list, the procedure echoes back the selection.

The procedure uses the INPUT FROM statement to read the contents of the user-specified directory and creates a comma-separated list of all the file and directory names in the directory. It then assigns the comma-separated list to the LIST-ITEMS attribute of the selection-list. Because an assignment to an attribute depends on the widget being located in a frame, the DEFINE FRAME statement is used to locate the selection-list.

r-select.p

```
DEFINE VARIABLE f-name        AS CHARACTER NO-UNDO FORMAT "x(14)".
DEFINE VARIABLE choice        AS CHARACTER NO-UNDO FORMAT "x(50)"
  LABEL "You have selected".
DEFINE VARIABLE list_contents AS CHARACTER NO-UNDO FORMAT "x(200)".
DEFINE VARIABLE dir           AS CHARACTER NO-UNDO FORMAT "x(40)"
  LABEL "Please enter a directory pathname ".
DEFINE VARIABLE sl            AS CHARACTER NO-UNDO
  VIEW-AS SELECTION-LIST INNER-CHARS 15 INNER-LINES 10 SORT.

DEFINE FRAME b sl.
DEFINE FRAME c choice.

DEFINE STREAM dirlist.

ENABLE dir WITH FRAME d WITH SIDE-LABELS.
ON RETURN OF dir IN FRAME d DO:
  ASSIGN FRAME d dir.
  INPUT STREAM dirlist FROM OS-DIR (dir).
  IMPORT STREAM dirlist f-name.
  list_contents = f-name.
  REPEAT:
    IMPORT STREAM dirlist f-name.
    list_contents = list_contents + "," + f-name.
  END.
  INPUT CLOSE.

  sl:LIST-ITEMS IN FRAME b = list_contents.
  ENABLE sl WITH FRAME b NO-LABELS TITLE "Please Select a File" WIDTH 50.
END.

ON VALUE-CHANGED OF sl IN FRAME b DO:
  choice = sl:SCREEN-VALUE.
  DISPLAY choice WITH FRAME c SIDE-LABELS.
END.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- When the selection-list appears, if it contains the value of the associated field or variable, that value is initially highlighted. Otherwise, no value in the selection-list is initially highlighted.
- The LIST-ITEMS option of the SELECTION-LIST phrase requires a list of quoted items ("a", "b", "c"), whereas the LIST-ITEMS attribute of a selection-list requires a quoted list of items ("a, b, c"). Similarly, the LIST-ITEM-PAIRS option of the SELECTION-LIST phrase requires a list of quoted items ("a", "1", "b", "2", "c", "3"); whereas the LIST-ITEM-PAIRS attribute of a selection-list requires a quoted list of items ("a, 1, b, 2, c, 3").
- If you specify the SORT option for a selection-list, then any items you add with ADD-FIRST, ADD-LAST, or INSERT methods are added in sorted order rather than the order you specify.
- In Windows, you can use a mnemonic to transfer focus to the selection-list.

See also

SIZE phrase, VIEW-AS phrase

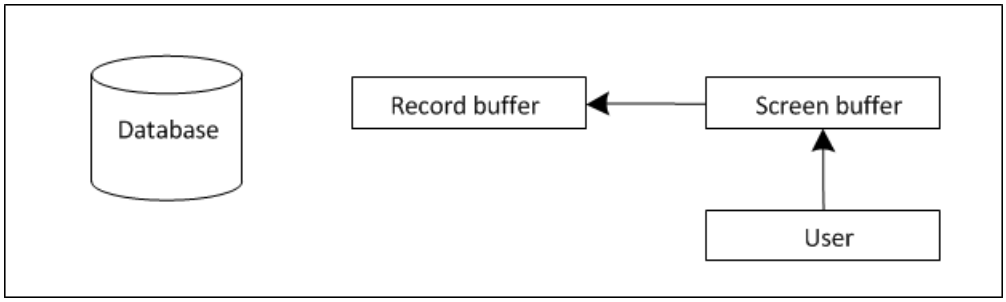
SET statement

Requests input, and then puts the data in the screen buffer frame and in the specified fields or variables. The SET statement is a combination of these statements:

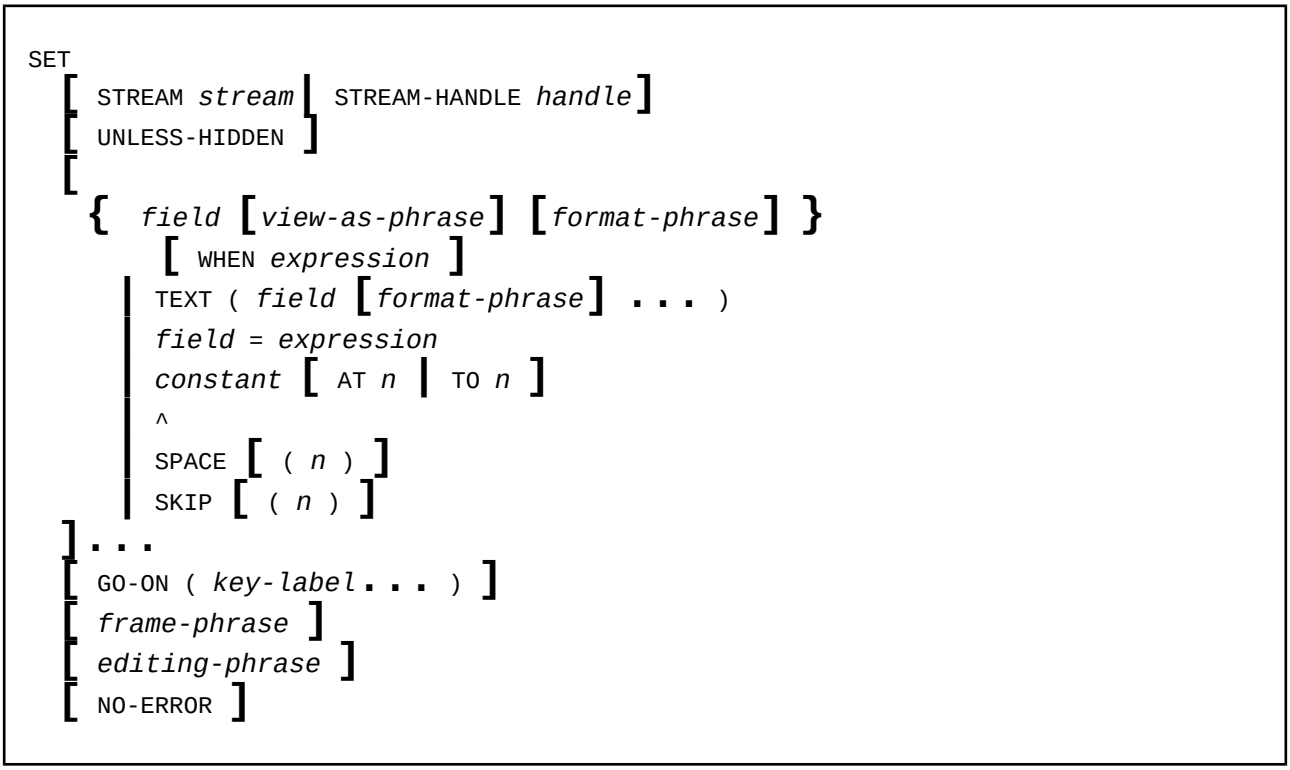
- **PROMPT-FOR** - Prompts the user for data and puts that data into the screen buffer
- **ASSIGN** - Moves data from the screen buffer to the record buffer

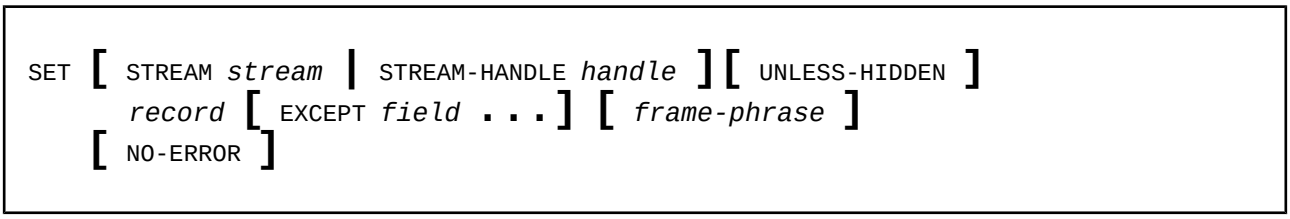
Note: Does not apply to SpeedScript programming.

Data movement



Syntax





`STREAM` *`stream`*

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream. See the DEFINE STREAM statement reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams.

`STREAM-HANDLE` *`handle`*

Specifies the handle to a stream. If *`handle`* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

`UNLESS-HIDDEN`

Restricts SET to fields whose HIDDEN attribute is FALSE.

`field`

Represents the name of the field or variable whose value you want to store in the screen buffer and in the field or variable.

In the case of array fields, array elements with constant subscripts are treated as any other field. Array fields with no subscripts are expanded as though you typed in the implicit elements. See the DISPLAY statement reference entry for information on how the AVM handles array fields with expressions as subscripts.

view-as-phrase

Specifies the widget used to represent the field. For more information on *view-as-phrase*, see the VIEW-AS phrase reference entry.

format-phrase

Specifies one or more frame attributes for a field, variable, or expression. For more information on *format-phrase*, see the Format phrase reference entry.

WHEN *expression*

Sets the field only when *expression* has a value of TRUE. An *expression* is a field name, variable name, or expression whose value is logical.

TEXT

Defines a group of character fields or variables (including array elements) to use automatic word wrap. The TEXT option works with character fields only. When you insert data in the middle of a TEXT field, the AVM wraps data that follows into the next TEXT field, if necessary. If you delete data from the middle of a TEXT field, the AVM wraps data that follows into the empty area. If you enter more characters than the format for the field allows, the AVM discards the extra characters. The character fields must be in the x(*n*) format.

A blank in the first column of a line marks the beginning of a paragraph. Lines within a paragraph are treated as a group and will not wrap into other paragraphs.

The following table lists the keys you can use within a TEXT field, and their actions.

Table 60: Key actions in a TEXT field

Key	Action
APPEND-LINE	Combines the line the cursor is in with the next line.
BACK-TAB	Moves the cursor to the previous TEXT field.
BREAK-LINE	Breaks the current line into two lines beginning with the character the cursor is on.
BACKSPACE	Moves the cursor one position to the left and deletes the character at that position. If the cursor is at the beginning of a line, BACKSPACE moves the cursor to the end of the previous line.
CLEAR	Clears the current field and all fields in the TEXT group that follow.
DELETE-LINE	Deletes the line the cursor is in.
NEW-LINE	Inserts a blank line below the line the cursor is in.
RECALL	Clears fields in the TEXT group and returns initial data values for the group.

Key	Action
RETURN	In overstrike mode, moves to the next field in the TEXT group on the screen. In insert mode, the line breaks at the cursor and the cursor is positioned at the beginning of the new line.
TAB	Moves to the field after the TEXT group on the screen. If there is no other field, the cursor moves to the beginning of the TEXT group.

In this procedure, the s-com, or Order Comments field is a TEXT field. Run the procedure and enter text in the field to see how the TEXT option works.

r-text.p

```
DEFINE VARIABLE s-com AS CHARACTER NO-UNDO FORMAT "x(40)" EXTENT 5.

FORM "Shipped   :" Order.ShipDate AT 13 SKIP
 "Misc Info :" Order.Instructions AT 13 SKIP(1)
 "Order Comments :" s-com AT 1
 WITH FRAME o-com CENTERED NO-LABELS TITLE "Shipping Information".

FOR EACH Customer NO-LOCK, EACH Order OF Customer:
 DISPLAY Customer.CustNum Customer.Name Order.OrderNum
   Order.OrderDate Order.PromiseDate WITH FRAME order-hdr CENTERED.
 UPDATE Order.ShipDate Order.Instructions TEXT(s-com)
   WITH FRAME o-com.
 s-com = "".
END.
```

field = expression

Indicates that the value of *field* is determined by evaluating the *expression* rather than having it entered on the screen or from a file. An assignment statement is embedded within the SET statement.

constant AT *n*

A constant value that you want to display in the frame. The *n* is the column in which you want to start the display.

constant TO *n*

A constant value that you want to display in the frame. The *n* is the column in which you want to end the display.

^

Tells the AVM to ignore an input field when input is being read from a file. Thus, the following statement reads a line from an input file and ignores that line:

```
SET ^
```

This is an efficient way to skip over lines.

SPACE [(*n*)]

Identifies the number (*n*) of blank spaces to insert after the expression on the display. The *n* can be 0. If the number of spaces you specify is more than the spaces left on the current line of the frame, the AVM starts a new line and discards any extra spaces. If you do not use this option or do not use *n*, the AVM inserts one space between items in the frame.

SKIP [(*n*)]

Identifies the number (*n*) of blank lines to be inserted after the expression is displayed. The *n* can be 0. If you do not use this option, a line is not skipped between expressions only if they do not fit on one line. If you use the SKIP option, but do not specify *n*, or if *n* is 0, a new line is started unless it is already at the beginning of a new line.

GO-ON (*keylabel . . .*)

Tells the AVM to take the GO action when the user presses any of the keys listed. The keys you list are used in addition to keys that perform the GO action by default or because of ON statements. When you list a key label in the GO-ON option, you use the keyboard label of that key. For example, if you want the AVM to take the GO action when the user presses **F1**, you use the statement GO-ON(**F1**). If you list more than one key, separate them with spaces, not commas, as in GO-ON(**F1RETURN**).

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

editing-phrase

Supported only for backward compatibility.

Identifies processing to take place as each keystroke is entered. This is the syntax for the *editing-phrase*:

```
[ label: ] EDITING: statement... END.
```

For more information on *editing-phrase*, see the EDITING phrase reference entry.

record

Represents the name of a record buffer. All of the fields in the record, except those with the data type RECID and ROWID, are processed exactly as if you set each individually. The record you name must contain at least one field.

To use SET with a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

EXCEPT *field*

Affects all fields except those fields listed in the EXCEPT phrase.

Note: Compound assignment operators (+=, -=, *=, /=) can be used with the SET statement. For more information, see += Addition assignment operator, -= Subtraction assignment operator, *= Multiplication assignment operator, and /= Division assignment operator.

Examples

The `r-set.p` procedure reads each Item record, displays the `ItemNum` and lets the user enter information for the `ItemName`, `OnHand`, `Allocated`, and `Price` fields. When you run this procedure, notice that it does not display existing values for the `ItemName`, `OnHand`, `Allocated`, and `Price` fields.

r-set.p

```
FOR EACH Item:
  DISPLAY Item.ItemNum.
  SET Item.ItemName Item.OnHand Item.Allocated Item.Price.
END.
```

On each iteration of the block, the FOR EACH statement reads a single record into the Item record buffer. The DISPLAY statement moves the `ItemNum` from the record buffer to the screen buffer where you can see it. The SET statement prompts for data, stores the data in screen buffers, and moves the data to the record buffer, overwriting whatever is already there. Therefore, even though the `ItemName`, `OnHand`, `Allocated`, and `Price` fields are put into the Item record buffer by the FOR EACH statement, you never see the values for those fields.

The `r-set2.p` procedure displays the `CustNum`, `Name`, and `CreditLimit` for a Customer and lets you change the `CreditLimit` field. The HELP option in the SET statement displays help information at the bottom of the screen when you are changing the `CreditLimit`. The VALIDATE option in the SET statement makes sure that the `CreditLimit` value is greater than 0. If it is not, VALIDATE displays the message "Invalid credit limit."

r-set2.p

```
FOR EACH Customer:
  DISPLAY Customer.CustNum Customer.Name Customer.CreditLimit.
  SET Customer.Name Customer.CreditLimit
    VALIDATE(Customer.CreditLimit > 0, "Invalid credit limit.")
    HELP "Enter a positive credit-limit.".
  REPEAT:
    CREATE Order.
    Order.CustNum = Customer.CustNum.
    SET Order.OrderNum Order.ShipDate VALIDATE(Order.ShipDate > TODAY,
      "Ship date too early.").
  END.
END.
```

After you modify `CreditLimit`, the procedure creates an Order for the Customer and assigns the `Customer.CustNum` value to the `CustNum` field in the Order record. The SET statement lets you enter information for the `OrderNum` and `ShipDate` fields. The VALIDATE option in the SET statement makes sure that the Ship Date is greater than TODAY.

Notes

- If any *field* is a field in a database record, the SET statement upgrades the record lock condition to EXCLUSIVE-LOCK before updating the record.
- If any *field* is part of a record retrieved with a field list, the SET statement rereads the complete record before updating it.
- SET does not move data into the field or variable if there is no data in the corresponding screen field. There is data in a screen field if a DISPLAY of the field was done or if you enter data into the field. If you set a field or variable that has not been DISPLAYed in the frame and key in blanks, then the field or variable is not changed because the screen field is changed only if the data differs from what was in the frame field.
- When ABL compiles a procedure, it designs all the frames used by that procedure. When you run the procedure, the SET statement puts data into those fields.
- In a SET statement, the AVM first prompts for all specified fields and then assigns the values of those fields, moving from left to right. During this left to right pass of the field list, the AVM processes embedded assignments (*field* = *assignment*) as it encounters them.
- If you are getting input from a device other than the terminal, and the number of characters read by the SET statement for a particular field or variable exceeds the display format for that field or variable, the AVM returns an error. However, if you are setting a logical field that has a format of y/n and the data file contains a value of YES or NO, the AVM converts that value to "y" or "n".
- If you type blanks into a field in which data has never been displayed, the ENTERED function returns FALSE and the SET or ASSIGN statement does not update the underlying field or variable. Also, if the AVM has marked a field as entered, and the SET statement prompts for the field again and you do not enter any data, the AVM no longer considers the field an entered field.
- If you use a single qualified identifier with the SET statement, the Compiler first interprets the reference as *dbname.tablename*. If the Compiler cannot resolve the reference as *dbname.tablename*, it tries to resolve it as *tablename.fieldname*.

  When using SET to set fields, you must use table names that are different from field names to avoid ambiguous references. See the Record phrase reference entry for more information.
- The SET statement causes the ASSIGN and WRITE events to occur and fires all related database ASSIGN and WRITE triggers. The ASSIGN triggers execute before the WRITE triggers and after the field is actually updated. The WRITE triggers only execute if the ASSIGN triggers do not return an error. If an ASSIGN trigger fails (or executes a RETURN statement with the ERROR option), the SET statement is undone. This means that any changes to the database from that SET statement are backed out. If the SET statement occurs within a transaction, any changes to variables, worktable fields, and temp-table fields are also undone unless they are defined with the NO-UNDO option. Also, if a WRITE trigger fails (or executes a RETURN statement with the ERROR option), the SET statement is undone. See *Use ABL Database Triggers and Indexes* for more information on database triggers.
- In the context of the .NET blocking method, `System.Windows.Forms.Application:Run( )`, if you execute the SET statement for a non-modal ABL window from within a user-defined function or non-VOID method, this statement raises the STOP condition. For more information on the .NET `Application:Run( )` method, see the reference entry for the WAIT-FOR statement (.NET and ABL).

See also

DEFINE STREAM statement, EDITING phrase, Format phrase, Frame phrase, NO-ERROR option PROMPT-FOR statement, Stream object handle, UPDATE statement

SET-BYTE-ORDER statement

Sets an internal indicator designating the byte-order of the data pointed to by the MEMPTR variable.

Note: Does not apply to SpeedScript programming.

Syntax

```
SET-BYTE-ORDER( memptr ) = integer-expression
```

memptr

An expression that returns a MEMPTR.

integer-expression

An expression that returns an integer value that will be used to indicate the byte-ordering of the data in the memory to which the MEMPTR points. *integer-expression* must be one of the reserved keywords defined in the following table or its corresponding value. If *integer-expression* is not valid, ABL generates an error.

Table 61: Byte order options

Keyword	Value	Description
HOST-BYTE-ORDER	1	Same format as the machine where the process that calls SET-BYTE-ORDER is running.
BIG-ENDIAN	2	A multiple-byte data type is stored with the high-order byte in the lowest address reserved for the data; successively lower-order bytes are stored at successively higher addresses. Note that Internet protocols use BIG-ENDIAN byte-ordering.
LITTLE-ENDIAN	3	A multiple-byte data type is stored with the low-order byte in the lowest address reserved for the data; successively higher-order bytes are stored at successively higher addresses.

Notes

- The byte order for a MEMPTR is HOST-BYTE-ORDER by default, that is, if SET-BYTE-ORDER has not been called for a given MEMPTR, its byte order is HOST-BYTE-ORDER.
- SET-BYTE-ORDER by itself never affects data currently in the MEMPTR, that is, it does not actually re-order the data. It only affects how subsequent calls to the PUT- statements and GET- functions work with that MEMPTR variable.

See also

GET-BYTE-ORDER function

SET-DB-CLIENT function

Sets the user identity for a specified connected OpenEdge database using an unsealed or a sealed client-principal object.

For an unsealed client-principal object (in the INITIAL state), this function performs a user authentication operation on the user identity asserted by the object. If successful, the function seals the client-principal and assigns the user identity to a specified database connection, and if the database is multi-tenant this also sets the user's tenancy.

For a sealed client-principal object (in the LOGIN state), this function performs a single sign-on (SSO) operation to validate the user identity represented by the object. If successful, the function assigns the user identity to a database connection, and if the database is multi-tenant this also sets the user's tenancy.

If the user identity is set successfully and the database is multi-tenant, the connection accesses the database through the tenant organization that is configured for the user's domain, and the tenant information is added to the client-principal object used to set the identity.

The function returns TRUE if the user identity is set successfully; if unsuccessful, the function returns FALSE, and the connection identity remains unchanged.

Note: Using this function overrides any user identity previously set for the database connection by either the SECURITY-POLICY:SET-CLIENT( ) method or the SETUSERID function.

Syntax

```
SET-DB-CLIENT ( { client-principal-handle | ? }
  [ , { integer-expression | logical-name |alias | ? } ] )
```

client-principal-handle

A handle to a client-principal object. If the client-principal object is unsealed, it must be initialized with the attribute values required by the SEAL( ) method in addition to any PRIMARY-PASSPHRASE attribute value required to authenticate the asserted user identity. If the object is sealed, it must be sealed with a domain access code that is the same as the access code configured for the user's domain stored in the domain registry trusted by the specified database connection. If this parameter is set to the Unknown value (?), the current identity of affected database connections remains unchanged, and the function unlocks and allows the connection identity to be set using the SET-CLIENT( ) method, as described for the setting of database parameters.

integer-expression | *logical-name* | *alias* | ?

Specifies a database connection on which to set the identity as follows:

integer-expression

The sequence number of a connected OpenEdge database for which to set the user identity. For example, `SET-DB-CLIENT(hCP, 1)` sets the user identity associated with the specified client-principal object for the first database connection, `SET-DB-CLIENT(hCP, 2)` sets the user identity for the second database connection, and so on. If you specify a sequence number that does not correspond to a connected database, the function returns FALSE.

logical-name | *alias*

The logical name or alias of a connected OpenEdge database for which to set the user identity. These forms require a quoted character string or a character expression. If you specify a logical name or alias that does not correspond to a connected database, the function returns FALSE.

If you pass a valid option to identify an OpenEdge database connection, the function operates on that OpenEdge database connection as follows:

- If the LOGIN-STATE attribute on the client-principal object is set to `"INITIAL"`, the function performs a user authentication operation, and if successful, sets user identity for the database connection,
- If the LOGIN-STATE attribute on the client-principal object is set to `"LOGIN"`, the function performs a validation (SSO) of the user identity, and if successful, sets the user identity for the database connection.
- If *client-principal-handle* is the Unknown value (?), the current connection identity remains unchanged, and the function removes the SECURITY-POLICY:SET-CLIENT( ) method lockout on the database connection.

If you do not pass an option to identify the specific database connection or you specify the Unknown value (?), the function operates on all OpenEdge database connections as follows:

- If the LOGIN-STATE attribute on the client-principal object is set to `"INITIAL"`, the function performs a user authentication operation for the first OpenEdge database connection, and if the authentication is successful, the function:
  1. Sets that database connection's user identity.
  2. With the client-principal LOGIN-STATE attribute set to `"LOGIN"`, performs an SSO operation to validate and (if successful) set the user identity for each remaining OpenEdge database connection in the ABL session.
- If the LOGIN-STATE attribute on the client-principal object is set to `"LOGIN"`, the function performs an SSO operation to validate and (if successful) set the user identity for each OpenEdge database connection in the ABL session.
- If *client-principal-handle* is the Unknown value (?), the current identity remains unchanged for all OpenEdge databases connections, and the function removes the SECURITY-POLICY:SET-CLIENT( ) method lockout on all the connections.

Example

To use the `_login.p` procedure that is provided with ABL, you must define user IDs and passwords for users who are authorized to access the database in user accounts associated with authentication-enabled domains. The following sample procedure (`r-login1.p`) is simplified from the `_login.p` procedure provided with ABL.

r-login1.p

```
/* Prompt user for userid and password and set the userid */
DEFINE VARIABLE id       AS CHARACTER FORMAT "x(255)" VIEW-AS FILL-IN
                            SIZE 50 BY 1 LABEL "User id" NO-UNDO.
DEFINE VARIABLE password AS CHARACTER FORMAT "x(255)" VIEW-AS FILL-IN
                            SIZE 50 BY 1 LABEL "Password" NO-UNDO.

DEFINE VARIABLE tries    AS INTEGER         NO-UNDO.
DEFINE VARIABLE lFound   AS LOGICAL         NO-UNDO.
DEFINE VARIABLE hCP      AS HANDLE          NO-UNDO.

CREATE CLIENT-PRINCIPAL hCP. /* create a CLIENT-PRINCIPAL only once during login*/

DO ON ENDKEY UNDO, RETURN:  /* return if they hit endkey */
  /* Reset id and password to blank in case of retry */
  ASSIGN
    id       = ""
    password = "".
  UPDATE SPACE(2) id SKIP  password BLANK
    WITH CENTERED ROW 8 SIDE-LABELS
    TITLE " Database " + LDBNAME("DICTDB") + " ".

  /* Use SET-DB-CLIENT instead of SETUSERID */
  hCP:INITIALIZE(id,?,?,password).

  IF NOT SET-DB-CLIENT(hCP,"DICTDB") THEN DO:
    MESSAGE "Sorry, userid/password is incorrect.".
    IF tries > 1 THEN QUIT. /* Only allow 3 tries */
    tries = tries + 1.
    UNDO, RETRY.
  END.

END.

DELETE OBJECT hCP.
hCP = ?.

HIDE ALL.
```

This procedure uses the SET-DB-CLIENT function to authenticate the prompted user identity passed in through the client-principal (hCP) to set the connection identity for the database with the logical name, `dictdb`.

If the value of the function is FALSE, the procedure allows the user another try. The user has three tries to log in. The first time, the `tries` variable is 0; `tries` is 1 the second time, and 2 the third. The third time, `tries` is greater than 1 and the procedure exits without any further attempts. If function value is TRUE, the database connection is set to the prompted user identity. In either case, the procedure exits after deleting the client-principal object and hiding the prompt.

Notes

- For any errors while operating on database connections, SET-DB-CLIENT returns FALSE, records any errors in the ERROR-STATUS system handle, and leaves the current identity for a given database connection unchanged. Also, a validation failure when setting the identity for a database connection does not stop SET-DB-CLIENT from operating on other connected databases. It records an error message for that database and attempts the operation on the next one.
- Within a transaction on a:
  - **Multi-tenant database** — Any attempt to set an identity for the connection that changes the current database tenancy raises a run-time error.
  - **Non-multi-tenant database** — As a best practice, Progress Software recommends that you **not** set a new identity for the connection.

- To assign a user identity while making a database connection, use the CONNECT statement.
- This function also checks the value of the LOGIN-EXPIRATION-TIMESTAMP attribute on the client-principal object. If the timestamp has expired before the function can seal (during user authentication) or validate (during SSO) the object, the function sets the LOGIN-STATE attribute to `"EXPIRED"` and returns FALSE without changing the current connection identity.
- To set a connection identity through a user authentication operation:
  - The client-principal object must be **unsealed** (in the INITIAL state).
  - The required attributes must be properly set (see the SEAL( ) method and PRIMARY-PASSPHRASE attribute entries).
  - The domain in the registry trusted by the database connection must be authentication-enabled. This can include a domain configured with a user-defined authentication system that has an ABL authentication plugin enabled. For information on OpenEdge support for domains and domain configuration, see *Learn about Identity Management*.

  When the user authentication operation fails for a database connection, for auditing purposes, this function implicitly executes the AUTHENTICATION-FAILED( ) method on the client-principal and leaves the previous connection identity unchanged.
- To set a connection identity through an SSO operation:
  - The client-principal object must be **sealed** and set to the LOGIN state either explicitly by the application (using the SEAL( ) method) or implicitly by OpenEdge.
  - The object must be valid according to the domain registry used by the database connection: it must be sealed using the access code defined for a registry domain whose name matches the domain name stored in the object.
  - The domain in the registry trusted by the database connection must be enabled and configured with an authentication system that supports SSO.
- To set a connection identity when the authentication system is your ABL application, you must manually authenticate the user credentials for the client-principal object, explicitly call the SEAL( ) method to seal the object, and perform an SSO operation using this function to set the connection identity.
- For an STS-enabled database, this function always checks the user identity against the database's own domain registry configuration.
- For databases not using STS, the domain registry this function uses to perform a user authentication or SSO operation depends on the database's own domain registry configuration. If it is configured to use the application domain registry, the function checks the user identity against the ABL session domain registry loaded using the SECURITY-POLICY:LOAD-DOMAINS( ) method. If it is configured **not** to use the application domain registry, the function checks the user identity against the database's own local domain registry that OpenEdge loads for the connection. For more information on domain and domain registry configuration, see *OpenEdge Getting Started: Identity Management*.
- After a user identity is set for a database connection, the AVM uses that identity to determine if the user has permission to access tables and fields in that particular database.
- If you do not specify a database or pass the Unknown value (?), the user identity that this function sets for all connected databases overrides the user identity previously set by a call to the SET-CLIENT( ) method.
- Any database connection whose user identity is set using this function locks out the SET-CLIENT( ) method from setting an identity for that connection until you use this function to unlock it.
- You can also use this function, instead of the SETUSERID function, to set the user identity for a database connection whether or not the user account is in the `_User` table.

- Calling this method generates an audit event, and creates an audit record for the event in all connected audit-enabled databases according to each database's current audit policy settings.
- This function returns FALSE and with messages recorded in the ERROR-STATUS system handle when:
  - A database connection is specified, the client-principal object is sealed, and SSO validation of its identity for the specified database connection fails.
  - No database connection is specified, the client-principal object is sealed, and SSO validation of its identity for any connected database fails. In this case, ERROR-STATUS records a message for each database connection that fails SSO validation.
  - The client-principal object is unsealed and the domain configuration is restricted to SSO operations.
  - A database connection is specified, the client-principal object is unsealed, and user authentication of its asserted identity on the specified database connection fails.
  - The client-principal object is sealed and the domain configuration is restricted to user authentication operations.
- This function does not attempt set the connection identity for the foreign data source of a DataServer connection. However, it does attempt to set the connection identity for the OpenEdge schema holder database.

See also

Client-principal object handle, SEAL( ) method, SET-CLIENT( ) method, SETUSERID function

---

SET-EFFECTIVE-TENANT function

Sets the effective tenancy of a multi-tenant database connection for a super-tenant user. A super tenant has no real tenancy of its own, but can act as if it is a particular regular tenant without the user having to authenticate a new connection identity that has that particular tenancy. Therefore, the *effective tenancy* is the acting regular tenancy of a super tenant.

Initially, OpenEdge sets the effective tenancy for any super-tenant identity to the default tenant. The purpose of this function is to change the effective tenancy of a super tenant identity to any regular tenant.

Syntax

```
SET-EFFECTIVE-TENANT ( tenant-expression [ , database-name ] )
```

tenant-expression

An integer or character string expression. If *tenant-expression* evaluates to an integer, the value must be a valid tenant ID for a regular or default tenant. If *tenant-expression* evaluates to a character string, the value must be a valid tenant name for a regular or default tenant. Otherwise, the statement raises ERROR.

database-name

A character expression that evaluates to a logical database name or database alias. If no database is specified and more than one database is connected, the AVM raises an error. If the database name is not a valid name for a connected database, the AVM raises an error.

Notes

- The function returns a logical TRUE if successful, and raises an error if not successful.
- If a regular-tenant user executes this function, the tenant identified by *tenant-expression* must match the user's tenancy; otherwise, the function raises an error. If a super-tenant user executes this function, and *tenant-expression* is not a valid regular tenant (not a super tenant), the function raises an error.
- All super-tenant users login with access to shared data and data owned by the effective default tenant, with access conditioned on the user's table and field permission settings. When a super-tenant user executes SET-EFFECTIVE-TENANT, they have access to shared data and data owned by the new effective tenant, with access, again, conditioned by the user's same table and field permission settings. This super-tenant access can include data for multiple tenants, **without** changing effective tenancy, by using the TENANT-WHERE option of the record phrase. (The TENANT-WHERE option, itself, does not change the effective tenancy.)
- When you execute SET-EFFECTIVE-TENANT, buffers and cursors for existing tenant data from a previous effective tenancy are not cleared. In this way, a super tenant can get records for more than one tenant. Note that regardless of changes in effective tenancy, any subsequent updates to prior tenant data are written to the original tenant tables.
- The scope of SET-EFFECTIVE-TENANT ends when a new identity is set for the database connection using the SET-DB-CLIENT function, the SECURITY-POLICY:SET-CLIENT( ) method, or the SETUSERID function, at which point the effective tenancy changes back to the default tenant for any super tenant user identity or to the real tenant of a regular tenant user identity (including the default tenant). Unlike setting the effective tenant, setting a new connection identity also clears buffers and cursors of any prior tenant data.
- An UNDO does not undo the tenancy of a SET-EFFECTIVE-TENANT.
- The AVM looks up *tenant-expression* in the `_Tenant` table with a share lock. The AVM waits 60 seconds to get the share lock, and if it fails to get the share lock in that amount of time, the AVM raises an error. The AVM releases the share lock immediately after successfully retrieving the tenant information. This share lock is released even if the function is called within a transaction.

See also

GET-EFFECTIVE-TENANT-ID function, GET-EFFECTIVE-TENANT-NAME function

SET-POINTER-VALUE statement

Sets a variable of type MEMPTR to the value of a particular memory location.

Note: Does not apply to SpeedScript programming.

Syntax

```
SET-POINTER-VALUE ( memptr-var ) = memptr-value
```

memptr-var

A reference to a variable defined as MEMPTR.

memptr-value

An integer that represents a memory location.

Example

The following example calls a DLL routine that returns a pointer to a structure, extracts an address at byte 5 of the structure, uses SET-POINTER-VALUE to assign the address to an ABL MEMPTR, and displays the character string at the address:

```
DEFINE VARIABLE person_struct AS MEMPTR NO-UNDO. /* pointer to structure */
DEFINE VARIABLE name          AS MEMPTR NO-UNDO. /* pointer to name */

SET-SIZE(person_struct) = 8.

RUN person_info (OUTPUT person_struct).
SET-POINTER-VALUE(name) = GET-LONG(person_struct,5).
DISPLAY GET-STRING(name,1) FORMAT "x(50)".
SET-SIZE(person_struct) = 0.

PROCEDURE person_info EXTERNAL "person.dll" PERSISTENT:
  DEFINE OUTPUT PARAMETER person_struct AS MEMPTR.
END PROCEDURE.
```

Notes

- SET-POINTER-VALUE is particularly useful when accessing Windows Dynamic Link Library (DLLs) or UNIX shared library routines from ABL. For more information on DLLs, see the topics on DLLs in *OpenEdge Programming Interfaces*.
- For more information on the MEMPTR data type, see *OpenEdge Programming Interfaces*.

See also

GET-POINTER-VALUE function, SET-SIZE statement

SET-SIZE statement

Manages memory associated with a MEMPTR variable. This includes allocating and associating a region of memory with an uninitialized MEMPTR variable, setting the size of a region allocated with a Windows dynamic link library (DLL) or UNIX shared library routine for a MEMPTR, and deallocating memory associated with a MEMPTR variable.

Note: Does not apply to SpeedScript programming.

Syntax

```
SET-SIZE ( memptr-var ) = size
```

memptr-var

A reference to a variable defined as MEMPTR.

size

An integer expression that specifies the allocated byte size of the region pointed to by *memptr-var*.

Example

In the following example, the SET-SIZE statement allocates 8 bytes of memory, associates the memory with the `ElipRegion` variable, and then initializes the region with four SHORT (2-byte) values:

r-setsiz.p

```
DEFINE VARIABLE ElipRegion AS MEMPTR NO-UNDO.

ASSIGN
 SET-SIZE(ElipRegion)     = 8
  PUT-SHORT(ElipRegion, 1) = 10
  PUT-SHORT(ElipRegion, 3) = 10
  PUT-SHORT(ElipRegion, 5) = 200
  PUT-SHORT(ElipRegion, 7) = 50.
```

Notes

- If *memptr-var* has no memory allocated to it (is uninitialized), then the SET-SIZE statement allocates a memory region of the specified size.
- If a MEMPTR variable is returned from a DLL or UNIX shared library routine that also allocates a memory region to it, then the SET-SIZE statement initializes the size of the existing region. The AVM does not allocate a new region. This allows the AVM to perform bounds checking on references to MEMPTR regions allocated outside ABL.

  Caution: You must know and specify the exact size of the memory region returned by the DLL routine from the type of structure it allocates. An incorrect size can result in data loss.

- If the specified size is 0, the SET-SIZE statement deallocates (frees) any memory associated with *memptr-var*, making it available to reference a new memory region.
- If the specified size is greater than 0 and *memptr-var* is fully initialized (associated with a memory region of a specified size), the SET-SIZE statement has no effect and leaves *memptr-var* unchanged.
- After initializing a MEMPTR variable, you can obtain the address of (or pointer to) the region associated with the variable using the GET-POINTER-VALUE function. Use this to build structures that contain pointers to other structures, as required by some DLL or UNIX shared library routines.
- For more information on accessing DLL routines from ABL, see *OpenEdge Programming Interfaces*.

See also

GET-POINTER-VALUE function, GET-SIZE function

SETUSERID function

Authenticates a user identity for a specified database connection, verifying that the user ID and password supplied to the SETUSERID function match a user account in the _User table of the database. (This restriction does not apply to authentication with an STS-enabled database. See notes below for more information.) If they match, the database connection is set to the specified user identity and the function returns a TRUE value. If the database is multi-tenant, is also sets the user's tenancy.

If the user ID is not in the _User table or the password is incorrect, SETUSERID returns a FALSE value and does not assign the user identity to the database connection. You can also check the ERROR-STATUS system handle for any messages returned.

Note:

Using this function overrides user identity previously set for the database connection by either the SECURITY-POLICY:SET-CLIENT( ) method or the SET-DB-CLIENT function.

This function authenticates user identities only against user accounts defined in the _User table of a connected OpenEdge RDBMS. To authenticate against additional OpenEdge-supported and application-defined authentication systems, use the SET-CLIENT( ) method or the SET-DB-CLIENT function, instead.

Syntax

```
SETUSERID ( userid , password [ , logical-dbname ] )
```

userid

A literal value, field name, variable name, or expression that results in a character value that represents the user's user ID. If you use a literal value, you must enclose it in quotation marks ("").

If the user is defined in a non-default domain, this value must be a fully qualified user ID, including both the non-qualified user ID and domain separated by a domain delimiter (@). If the user is defined in the default (blank) domain, only a non-qualified user ID (without a domain delimiter) must be specified. For more information on specifying a fully qualified user ID, see the reference entry for the QUALIFIED-USER-ID attribute.

Note: The default blank domain provides backward compatibility with OpenEdge releases that do not support domains as part of a user's identity.

password

A literal value, field name, variable name, or other character expression that results in a character value that represents the user's password. If you use a literal value, you must enclose it in quotation marks ("").

Alternatively, you can encrypt the password using the ENCRYPT-AUDIT-MAC-KEY( ) method. For more information, see the topics on encrypted passwords in *OpenEdge Programming Interfaces*.

logical-dbname

The logical name of the database on whose connection you want to check and set the user identity. The logical database name must be a character string enclosed in quotes, or a character expression.

If you do not specify this argument, the compiler inserts the name of the database that is connected when the procedure is compiled. If you omit this argument and more than one database is connected, ABL raises an error.

Notes

- Within a transaction on a:
  - **Multi-tenant database** — Any attempt to set an identity for the connection that changes the current database tenancy raises a run-time error.
  - **Non-multi-tenant database** — As a best practice, Progress Software recommends that you **not** set a new identity for the connection.
- To assign a user identity while making a database connection, use the CONNECT statement.
- After a user identity is set for a database connection, the AVM uses that identity to determine if the user has permission to access tables and fields in that particular database.
- This function has the following restrictions:
  - It does not generate any audit events, such as for login and logout.
  - It can change the user identity for only one database connection at a time.
  - The domain of the user identity you want to set must be configured to use the authentication system, `_oeusertable`.
  - The domain of the user identity you want to set must be enabled in the `_User` table of the connected database.

Note: When using an STS-enabled database, the `_oeusertable` authentication system is not required, and accounts do not have to be in the `_User` table.

- Under the following conditions, the SETUSERID function returns a value of FALSE and does not assign a user identity to the user:
  - There are no entries in the `_User` table.
  - There is no `_User` record with the same user ID as the one supplied with the SETUSERID function.
  - The password supplied with the SETUSERID function does not match the password in the `_User` table record of the specified user ID.
- When using the SETUSERID function, ABL returns a compiler error under the following conditions:
  - There is no database connected.
  - The *`logical-dbname`* argument is omitted, and more than one database is currently connected.
- When specifying the *`logical-dbname`* argument, you must provide the name of the logical database, not the physical database.
- SETUSERID encodes the *`password`* argument and then compares the result with the value stored in the `_User._password` field of the `_User` table.
- After SETUSERID returns a value of TRUE and assigns the authenticated user ID to a database connection:
  - ABL uses that user ID when the user compiles procedures.
  - Subsequent invocations of the USERID function for the same database connection return the assigned user ID.

- If the root user ID does not exist in the `_User` table, SETUSERID returns a value of FALSE when supplied with a userid of root. If the `_User` table does have a root entry, the user who assumes that user ID has all the privileges associated with the root user ID on UNIX.
- You must create and enable a blank user ID ("") if you want to set the user ID to a null value.
- See *Learn about Identity Management*, *OpenEdge Programming Interfaces*, and *Manage the OpenEdge Database* for more information on user privileges.
- Once an initial database connection is established, you can also use the SET-CLIENT( ) method (on the SECURITY-POLICY system handle) or the SET-DB-CLIENT function to set the user identity for the connection.
- Any database connection whose user identity is set using this function locks out the SET-CLIENT( ) method from setting an identity for that connection. To unlock and allow the SET-CLIENT( ) method to set the identity for a locked database connection, call the SET-DB-CLIENT function for the connection, passing the Unknown value (?) for its *`client-principal-handle`* parameter. This unlocking of a database connection leaves the current connection identity unchanged.
- For each user identity authenticated by the SETUSERID function, the function creates a sealed security token containing the user credentials for the database connection, which you can return as a client-principal object using the GET-DB-CLIENT function. You can then use this sealed client-principal in a single sign-on (SSO) operation to set the identity of another database connection or ABL session.
- This function returns FALSE and with messages returned in the ERROR-STATUS system handle if:
  - The format of the non-qualified user ID or domain name is invalid or the value has contains characters outside of the restricted user ID or domain name character sets (see the QUALIFIED-USER-ID attribute entry)
  - Multiple databases are connected and *`logical-dbname`* is not passed or resolves to the Unknown value (?)
  - Multiple databases are connected and *`logical-dbname`* does not refer to a connected database
- This function does not attempt set the connection identity for the foreign data source of a DataServer connection. However, it does attempt to set the connection identity for the OpenEdge schema holder database.

See also

Client-principal object handle, CONNECT statement, ENCRYPT-AUDIT-MAC-KEY( ) method, SET-CLIENT( ) method, SET-DB-CLIENT function, USERID function

SHA1-DIGEST function

Hashes the specified data using the United States Government Secure Hash Algorithm (SHA-1), and returns a 20-byte binary message digest value as a RAW value.

Syntax

```
SHA1-DIGEST( data-to-hash [ , hash-key ] )
```

data-to-hash

The source data to hash. The data may be of type CHARACTER, LONGCHAR, RAW, or MEMPTR. If the data is a CHARACTER or LONGCHAR value, the AVM converts it to UTF-8 (which ensures a consistent value regardless of code page settings). To avoid this automatic conversion, specify a RAW or MEMPTR value.

hash-key

An optional key value to use in the hash operation. The key may be of type CHARACTER, LONGCHAR, RAW, or MEMPTR. If the key is a CHARACTER or LONGCHAR value, the AVM converts it to UTF-8 (which ensures a consistent value regardless of code page settings). To avoid this automatic conversion, specify a RAW or MEMPTR value. This key value is combined with the source data before the hash operation begins.

If the *hash-key* value contains a null character, the null character is included in the hash operation.

Note: This function is not compatible with HMAC-SHA-1.

See also

MD5-DIGEST function, MESSAGE-DIGEST function

SHOW-STATS statement

Writes procedure call statistics to the `proc.mon` output file if you specify the Statistics with Cross-Reference (`-yx`) parameter. It also writes procedure access and usage statistics to the `client.mon` output file if you specify the Statistics (`-y`) parameter, Statistics with **CTRL+C** (`-yc`) parameter, Segment Statistics (`-yd`) parameter, or Statistics with Cross-Reference (`-yx`) parameter. If you specify Segment Statistics (`-yd`), it also displays statistics for each code segment.

Ordinarily, when you specify these startup parameters, the AVM writes the statistics to the output files at the end of your ABL session. This might not be what you want. For example, if you start ABL using the `-y` or `-yc` parameters, you might want to view the execution buffer statistics as they occur during your ABL session. With SHOW-STATS, you can force the AVM to write the statistics at a specific time, instead of at session end. For more information on these startup parameters, see *Startup Command and Parameter Reference*.

The SHOW-STATS statement also writes the value of the STARTUP-PARAMETERS attribute to the `client.mon` output file.

Syntax

```
SHOW-STATS [ CLEAR ]
```

CLEAR

Resets all counters and timers that the AVM uses to monitor the procedure call, procedure access, and usage statistics.

Example

This procedure runs the Data Dictionary and writes the procedure call, procedure access, and usage statistics to the `proc.mon` and `client.mon` output files:

r-stats.p

```
RUN dict.p.
SHOW-STATS.
```

Notes

- If you use the SHOW-STATS statement without specifying the Statistics (y) parameter, the AVM opens the `client.mon` file as if you were dynamically specifying -y. However, the first SHOW-STATS statement that you use does not send any statistics to the `client.mon` file; it only opens the file. All subsequent SHOW-STATS statements, however, send procedure access and usage statistics to the file. But since you did not specify -y at startup, the AVM does not write any startup parameter statistics to the `client.mon` file.
- You must specify the Statistics with Cross-Reference (-yx) parameter, if you want the SHOW-STATS statement to write procedure call statistics to the `proc.mon` file.

SIZE phrase

Specifies the width and height of a widget. You can express the dimensions in either character units or pixels.

Syntax

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

{ SIZE | SIZE-CHARS }

Specifies that the unit of measure is characters.

`SIZE-PIXELS`

Specifies that the unit of measure is pixels.

`width`

Specifies the width of the widget. If the units are characters, *`width`* must be a decimal constant. If the units are pixels, *`width`* must be an integer constant.

`height`

Specifies the height of the widget. If the units are characters, the value *`height`* must be a decimal constant. If the units are pixels, *`height`* must be an integer constant.

Example

The following example uses SIZE phrases to set the initial dimensions of the rectangle rec and to set the dimensions of the frame sz-frame. When you choose the b_size button, the rectangle is randomly resized.

r-size.p

```
DEFINE BUTTON b_quit LABEL "Quit"
  TRIGGERS:
    ON CHOOSE QUIT.
  END.

DEFINE BUTTON b_size LABEL "Size It".
DEFINE RECTANGLE rec SIZE 5 BY 5.
DEFINE FRAME butt-frame
  b_size b_quit
  WITH CENTERED ROW SCREEN-LINES - 2.

DEFINE FRAME sz-frame
  SKIP(1) SPACE(1) rec
  WITH SIZE 80 BY 10 TITLE "The rectangle is 5 by 5".

ON CHOOSE OF b_size IN FRAME butt-frame
  ASSIGN
    rec:WIDTH-CHARS IN FRAME sz-frame =
      RANDOM(1, FRAME sz-frame:WIDTH-CHARS - 3)
    rec:HEIGHT-CHARS     = RANDOM(1, FRAME sz-frame:HEIGHT-CHARS - 2)
    FRAME sz-frame:TITLE = "The rectangle is " + STRING(rec:WIDTH-CHARS) +
      " by " + STRING(rec:HEIGHT-CHARS).ENABLE rec WITH FRAME sz-frame.

ENABLE b_size b_quit WITH FRAME butt-frame.
WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame.
```

Notes

- ABL supports fractional character units. Therefore, if you express dimensions in characters, the width and height values can include up to two decimal places.
- For SpeedScript, the PIXEL options are not valid.

See also

COMBO-BOX phrase, DEFINE BROWSE statement, DEFINE BUTTON statement, DEFINE IMAGE statement, DEFINE RECTANGLE statement, EDITOR phrase, Frame phrase, RADIO-SET phrase, SELECTION-LIST phrase, SLIDER phrase

SLIDER phrase

Describes a slider representation of a field or variable. A slider is a graphical representation of a numeric range. It is composed of a rectangular area that contains a line or trackbar. A marker or pointer within the region indicates the current value. The SLIDER phrase is an option of the VIEW-AS phrase.

Note: Does not apply to SpeedScript programming.

Syntax

```
VIEW-AS SLIDER
  MAX-VALUE max-value MIN-VALUE min-value
 [ HORIZONTAL | VERTICAL ]
 [ NO-CURRENT-VALUE ]
 [ LARGE-TO-SMALL ]
 [ TIC-MARKS { NONE | TOP | BOTTOM | LEFT | RIGHT | BOTH }
   [ FREQUENCY n ] ]
 [ TOOLTIP tooltip ]
 [ size-phrase ]
```

`MAX-VALUE` *`max-value`* `MIN-VALUE` *`min-value`*

Sets the range of values for the slider. Both *`max-value`* and *`min-value`* must be integer constants. Depending on the windowing system in use, the maximum value, minimum value, or both can be displayed with the slider. If you do not specify either a minimum value or a maximum value, the default is 0. *`Max-value`* must be greater than *`min-value`*.

In Windows only, you can use the MAX-VALUE and MIN-VALUE options with the LARGE-TO-SMALL option to indicate that the slider's maximum display value displays first and the minimum value displays last as you move the slider control.

`HORIZONTAL | VERTICAL`

Specifies the orientation of the slider. If the orientation is VERTICAL, the slider displays with the minimum value at the bottom and the maximum value at the top. The user can then change the value by moving the trackbar up or down. If the orientation is HORIZONTAL (the default), the slider displays with the minimum value at the left and the maximum value at the right. The user can then change the value by moving the bar left or right.

`NO-CURRENT-VALUE`

The default is to display the current value for a given position on the slider control. The NO-CURRENT-VALUE option allows you to override this default behavior to indicate that the slider will not automatically display the current value of the slider.

For example, if the MIN-VALUE is 10, the default is to display the value 10 when the slider is first realized, and to update the displayed value whenever a user moves the slider trackbar.

The NO-CURRENT-VALUE option is supported in Windows only.

`LARGE-TO-SMALL`

The default numeric range that a slider can display is small (minimum) to large (maximum). The LARGE-TO-SMALL option allows you to override this default behavior as follows:

- When the slider is positioned horizontally, the left-most position on the trackbar displays the maximum value and the right-most position displays the minimum value.
- When the slider is positioned vertically, the bottom-most position on the trackbar displays the maximum value and the top-most position displays the minimum value.

The LARGE-TO-SMALL option is supported in Windows only.

TIC-MARKS { NONE | TOP | BOTTOM | LEFT | RIGHT | BOTH }

Enables the display of short hash marks on the outside of a slider to help indicate the movement of the trackbar with the slider widget. The default is not to display tic marks. If you specify the TIC-MARKS option, it is assumed that you are using new code to create a slider, and the trackbar on the slider widget will be relatively large.

However, if you leave the TIC-MARKS option out, ABL assumes that you are migrating old code, and the default size of the slider is the size originally defined for the slider in the old code.

If you want to use the large trackbar but do not want tic marks to display, specify TIC-MARKS NONE.

To implement the TIC-MARKS option, you must also specify on which side, or sides, of the trackbar tick-marks display by using the additional TOP, BOTTOM, LEFT, RIGHT, or BOTH qualifying options.

The TIC-MARKS option is supported in Windows only.

FREQUENCY *n*

Used only with the TIC-MARKS option, indicates the incremental display of the TIC-MARKS. For example, if you indicate a frequency of 5, a tic mark displays in every fifth position along the slider bar.

The FREQUENCY option is supported in Windows only.

TOOLTIP *tooltip*

Allows you to define a help text message for a text field or text variable. The AVM automatically displays this text when the user pauses the mouse pointer over a text field or text variable for which a ToolTip is defined.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the ToolTip is removed. No ToolTip is the default. The TOOLTIP option is supported in Windows only.

size-phrase

Specifies the outside dimensions of the slider widget. This is the syntax for *size-phrase*:

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

For more information, see the SIZE phrase reference entry.

Example

The following procedure displays a slider with tic-marks noted every tenth position, and prompts the user to pick an integer value. After the user picks an integer, the program displays in a separate frame the text "You selected" followed by the value.

r-slide.p

```
DEFINE VARIABLE choice AS INTEGER NO-UNDO LABEL "You selected".
DEFINE VARIABLE a      AS INTEGER NO-UNDO.

UPDATE a VIEW-AS SLIDER MAX-VALUE 100 MIN-VALUE 1 SIZE-CHAR 33 BY 3
  TIC-MARKS BOTTOM FREQUENCY 10
  LABEL "Slide to select an integer. Then press GO."
  WITH FRAME f Three-D.

choice = a.

DISPLAY choice WITH FRAME b SIDE-LABELS THREE-D.
PAUSE.
```

Notes

- If the slider is too short, the user might not be able to select from the full range of values.
- If you display the slider horizontally, the *width* value determines the length of the slider, and the *height* value adds white space above and below the slider; similarly, if you display the slider vertically, the *height* value determines the length of the slider, and the *width* value adds white space on either side of the slider.
- Note that Windows allows a user to transfer focus to the slider by pressing **ALT** and one of the letters in the label.
- In character interfaces, a slider widget has a minimum width that is dependent on the specified maximum value (MAX-VALUE attribute). The minimum height for a slider widget in a character interface is 2 character units. You can specify a value as low as 1.5 character units for the height of a slider in a character interface; however, ABL rounds the value up to 2 character units.

See also

VIEW-AS phrase

SQRT function

Returns the square root (as a DECIMAL value) of an expression you specify.

Syntax

```
SQRT ( expression )
```

expression

A numeric expression. If the value of the expression is negative, SQRT returns the Unknown value (?).

Example

This procedure prompts for a number and then displays the square root of that number:

r-sqrt.p

```
DEFINE VARIABLE iNum AS INTEGER NO-UNDO FORMAT ">,>>9"
  LABEL "Enter a number between 1 and 9,999".

REPEAT WITH SIDE-LABELS CENTERED
  TITLE "SQUARE ROOT GENERATOR" COLUMN 20 1 DOWN:
  DISPLAY SKIP(2).
  SET iNum SKIP(2).
  DISPLAY "The square root of " + STRING(iNum) + " is" FORMAT "x(27)"
     SQRT(num) FORMAT ">>>.9999".
END.
```

SSL-SERVER-NAME function

Returns the digital certificate subject name for an OpenEdge database connected via TLS. If a database connection does not exist or the connection is not using TLS, this function returns the Unknown value (?).

Syntax

```
SSL-SERVER-NAME ( logical-database-name )
```

logical-database-name

A quoted character string or character expression that specifies the database by its logical name.

Example

The following example returns the digital certificate subject name of the database with the logical name `mydb`:

```
SSL-SERVER-NAME (mydb).
```

See also

Server object handle, Socket object handle

STATUS statement

Specifies the text that appears in the status line of a window. The AVM displays the following default messages on that line:

- When a procedure is blocked and is waiting for the user to enter data into a frame field, the status message is "Enter data or press *end-error* to end," where *end-error* is the key label for the **END-ERROR** key.
- When a procedure reaches a PAUSE statement, the status message is "Press space bar to continue."
- While a procedure is not blocked for input, the status message is blank.

Note: Does not apply to SpeedScript programming.

Syntax

```
STATUS
  {   DEFAULT [ expression ]
    | INPUT [ OFF | expression ]
  }
  [ IN WINDOW window ]
```

`DEFAULT` *`expression`*

Replaces the default status message when a user is running a procedure (the default status message is blanks). The *`expression`* must be character and must be enclosed in quotes if it is a constant. If you do not specify an *`expression`*, the AVM resets the STATUS DEFAULT line to its original state. The STATUS DEFAULT is a maximum of 63 characters.

`INPUT OFF`

Tells the AVM not to display an input status message.

`INPUT` *`expression`*

Replaces the default status message when a user is entering data into a frame field. The *`expression`* must be character and must be enclosed in quotes if it is a constant. If you do not specify an *`expression`*, the AVM resets the STATUS INPUT line to its original state.

`IN WINDOW` *`window`*

Specifies the window in which to set the status message. If you omit the IN WINDOW phrase, the STATUS statement applies to the current window.

Example

This procedure replaces the default status messages with two other messages:

r-status.p

```
STATUS DEFAULT "All Around Sports Order Processing System".
 \STATUS INPUT "Enter data, or use the " + KBLABEL("END-ERROR") +
  " key to exit".

FOR EACH Customer:
  DISPLAY Customer.Name.
  FOR EACH Order OF Customer:
    UPDATE Order.OrderNum Order.PromiseDate Order.OrderDate Order.ShipDate.
  END.
  UPDATE Customer.CreditLimit.
END.
```

Notes

- After you use the STATUS DEFAULT, STATUS INPUT OFF, or STATUS INPUT statement during a session, that statement is in effect for all the procedures run in that session, unless it is overridden by other STATUS statements in those procedures, or until you return to the Procedure Editor.
- You cannot use the STATUS statement to change the default status messages displayed while you are in the Procedure Editor.
- You can use the PAUSE statement to override the default status message displayed when the AVM encounters a PAUSE statement.
- When you use the HELP attribute to display help text for a widget, the AVM overwrites the status text with the HELP text.

See also

MESSAGE statement, PAUSE statement

STOP statement

Signals the STOP condition and throws an associated `Progress.Lang.Stop` object in the current block. By default, the STOP condition stops processing a procedure, backs out the active transaction, and unwinds the call stack until it returns to the startup procedure or the Procedure Editor. You can change this behavior by including the ON STOP phrase on a block statement or by catching the associated `Stop` object using a CATCH block.

Syntax

```
STOP
```

Example

In any procedure, the outermost block that updates the database is the system transaction. In this procedure, the first iteration of the FOR EACH block starts a system transaction. The transaction ends when that iteration ends. Another transaction starts at the start of the next iteration. After you update the credit-limit field, the AVM prompts you to STOP. If you enter yes, the STOP statement stops the procedure and undoes any database modifications made in that transaction, as shown:

r-stop.p

```
DEFINE VARIABLE ans AS LOGICAL NO-UNDO.

FOR EACH Customer:
  DISPLAY Customer.CustNum Customer.Name.
  UPDATE Customer.CreditLimit.
  ans = FALSE.
  MESSAGE "Stopping now undoes changes to this record.".
  MESSAGE "Do you want to stop now?" UPDATE ans.
  IF ans THEN STOP.
END.
```

When you add the ON STOP phrase to the block statement of the previous procedure, it changes the default behavior of the STOP statement. In this procedure, the AVM allows you to re-enter the record when you choose to stop:

r-stop2.p

```
DEFINE VARIABLE ans  AS LOGICAL NO-UNDO.

FOR EACH Customer ON STOP UNDO, RETRY:
  DISPLAY Customer.CustNum Customer.Name.
  UPDATE Customer.CreditLimit.
  ans = FALSE.
  MESSAGE "Stopping now undoes changes to this record."
    "Do you want to stop now?"
    VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO UPDATE ans.
  IF ans THEN STOP.
END.
```

Notes

- Unless you coded an ON STOP phrase or CATCH block to trap the STOP condition, the STOP statement continues to back out of all blocks on the call stack.
- Almost all STOP conditions are trappable with the ON STOP phrase or a stop CATCH block. In some cases, the AVM might ignore STOP conditions at certain levels of the call stack. For example, if the AVM executes a procedure that relies on a lost database connection, the AVM raises the STOP condition and unwinds the call stack until it gets to a level **above** all references to the lost database. If the AVM encounters the ON STOP phrase or a compatible stop CATCH block before this point, the AVM ignores the phrase or CATCH block. If the AVM encounters the ON STOP phrase or a compatible stop CATCH block after this point, the AVM executes the directive specified in the phrase or the CATCH block, with the CATCH block taking precedence if the ON STOP phrase is on the same associated block.
- If you use the Startup Procedure (`-p`) parameter to start the ABL session, and if the startup procedure is still active, the default STOP action restarts the procedure.
- A terminal user can initiate the STOP condition by pressing **STOP**. This is usually mapped to **CTRL+BREAK** (Windows) or **CTRL+C** (UNIX). The actual mapping depends on your terminal and system configuration.

See also

Progress.Lang.Stop class

STOP-AFTER phrase

The `STOP-AFTER` phrase specifies a time-out value for a `DO`, `FOR`, or `REPEAT` block.

Syntax

```
STOP-AFTER time-limit
```

`time-limit`

An integer expression which specifies the number of seconds each iteration of a block has until a time-out occurs.

If a time-out occurs, the AVM raises the STOP condition on the current statement and throws a Progress.Lang.StopAfter object. Use an ON STOP phrase on the block (or an enclosing block) to alter the default STOP condition handling. Alternatively you can catch a Progress.Lang.StopAfter object using a CATCH block to provide alternate STOP condition handling. For more information, see the Progress.Lang.StopAfter class.

The STOP-AFTER time limit is the maximum time allowed for a single block iteration. For example, if you have STOP-AFTER 3 on a FOR EACH statement, each loop iteration must complete within 3 seconds or a STOP condition is raised.

If a block with a STOP-AFTER phrase contains a nested block with a STOP-AFTER phrase, then each is governed by it's own time limit. If the outer block time limit is reached while the inner block is executing, the STOP condition is raised even if the time limit for the inner block has not been reached.

If the STOP condition is handled and execution resumes within the scope of a block with a STOP-AFTER phrase, the time limit is not in effect until the next iteration of the block

If *time-limit* evaluates to zero or less, this is the equivalent of not specifying a STOP-AFTER phrase.

Examples

Example 1

The following example shows how the STOP-AFTER phrase can be used for placing a time limit on a query.

```
FOR EACH Customer WHERE <insert complicated query here> STOP-AFTER 5:
   <ABL statements>
   CATCH eStopAfter AS PROGRESS.Lang.StopAfter:
      MESSAGE "Query time limit exceeded" VIEW-AS ALERT-BOX.
   END.
END.
```

Example 2

The following example demonstrates how a STOP-AFTER phrase on an outer block also limits the time allowed for an inner block.

```
DEFINE VARIABLE ctr AS INTEGER INITIAL 0.

DO STOP-AFTER 5 ON STOP UNDO, LEAVE:

   MESSAGE "Start of DO BLOCK. Time is: " STRING(TIME,"HH:MM:SS").

   FOR EACH Customer STOP-AFTER 10:
      REPEAT:    // infinite loop
         ctr = ctr + 1.
      END.
      MESSAGE "After REPEAT block.".  // this message will never display
   END.

   MESSAGE "After FOR EACH block.".  // this message will never display

   CATCH eStopAfter AS Progress.Lang.StopAfter:
       MESSAGE "Time limit reached. Time is: " STRING(TIME,"HH:MM:SS") " ctr = " ctr

          VIEW-AS ALERT-BOX.
   END CATCH.

END.

MESSAGE "Procedure complete."
```

If you run this code, the outer DO block establishes a 5 second time limit for the work of the DO block and **all** inner blocks.

When the inner FOR EACH block starts, a separate time limit is set for each iteration of this block due to the STOP-AFTER 10 phrase.

Execution then continues to the REPEAT block which is an infinite loop. At some point, the outer 5 second time limit of the DO block is reached and the AVM raises the STOP condition. The STOP condition is raised on the statement the AVM is executing when the time limit is reached. Normal STOP handling proceeds from that point.

Because a CATCH statement on a block takes precedence over an ON STOP phrase, the STOP condition is caught and the "Time limit reached..." message displays. Notice that 5 seconds elapse from the "Start of DO BLOCK..." message to the "Time limit reached..." message.

Normal execution resumes following the DO block, and the "Procedure complete." message displays.

Notes

- STOP-AFTER phrases are not intended to interact with user interfaces.
- Blocking calls to third party software components, where the AVM has transferred execution control, cannot be timed out. This category includes operating system calls, MS Windows system calls, and calls to any third party DLLs and Unix shared objects.

See also

CATCH statement, DO statement, FOR statement, ON STOP phrase, Progress.Lang.StopAfter class, REPEAT statement

---

STRING function

Converts a value of any data type into a character value.

Syntax

```
STRING ( source [ , format ] )
```

source

An expression of any data type that you want to convert to a character value.

format

The format you want to use for the new character value. This format must be appropriate to the data type of *source*. If you do not use this argument, ABL uses the EXPORT format for all data types (except DATETIME and DATETIME-TZ, in which case it uses the default display format). This is useful if you want to produce left-justified numbers. For information on data display formats, see *ABL Essentials*.

Example

In the example procedure, the TIME function returns the number of seconds since midnight. The first DISPLAY statement in this procedure uses the STRING function to convert that value into hours and minutes. TIME is the value and "HH:MM AM" is the format used to display the result of the STRING function.

The second DISPLAY statement displays some Customer information. It uses the concatenation (+) operator to join together the values of the `City`, `State`, and `PostalCode` fields. If these fields were not joined together, the spacing would be different for each Customer address depending on the length of the city name.

r-string.p

```
DISPLAY SKIP(2) "The time is now" STRING(TIME,"HH:MM AM") SKIP(2)
  WITH NO-BOX NO-LABELS CENTERED.
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.Name + "  --" +
    STRING(Customer.CustNum, ">>>9") FORMAT "x(30)" AT 1
    Customer.Address AT 33
    Customer.City + ", " + Customer.State + " " + Customer.PostalCode
      FORMAT "x(22)" AT 33 SKIP(1)
    WITH NO-BOX NO-LABELS CENTERED.
END.
```

When you concatenate character fields, the AVM creates a new character field, at least for the duration of the procedure. The default display format for character expressions such as that resulting from the concatenation is x(8). This means that the AVM allows only 8 spaces for displaying the concatenation of the `City`, `State`, and `PostalCode` fields. The FORMAT x(22) option overrides that default x(8) format, telling the AVM to set aside 22 spaces for displaying the concatenation of the `City`, `State`, and `PostalCode` fields.

Notes

- The STRING function is double-byte enabled. The *`source`* argument can contain double-byte data.
- If *`source`* is an integer and *`format`* begins HH:MM or HH:MM:SS, STRING formats the *`source`* as a time. If the hour is greater than or equal to 12 and there is an A or an a in *`format`*, STRING subtracts 12 from the hour and converts the A or the a to a P or p (for A.M. and P.M.). The hour 0 is treated as 12 a.m., and noon is treated as 12 p.m. If you use AM/PM format, HH is replaced by a leading blank and a digit if the hour is between 0 and 9.

  If seconds (SS) are not in the format, then the time is truncated to hours and minutes.
- If *`source`* is a RAW value, you must specify an appropriate *`format`* to return the character string representation.
- When *`source`* is a DATETIME or DATETIME-TZ expression, the STRING function converts the expression to a character value in the specified format. If *`source`* is a DATETIME expression, and a time zone offset is present in the format string, the character value contains the time zone offset of the session. If *`source`* is a DATETIME-TZ expression, and time zone offset is not present in the format string, the character value contains the local date and time relative to the time zone of the DATETIME-TZ value.
- The STRING function converts a DATE, and the date part of a DATETIME or DATETIME-TZ, using the format specified by the DATE-FORMAT attribute or the Date Format (`-d`) startup parameter.

  For more information about the Date Format (`-d`) startup parameter, see *Startup Command and Parameter Reference*.
- You can use the STRING function to convert an object reference for a class instance to a character value. The STRING function implicitly calls the ToString( ) method (Object) of the class to convert the specified object reference.

- When applied to an enum instance, the STRING function implicitly calls the ToString( ) method (Object), which returns the name of the enumeration member, or a comma-delimited list of members in the case of a flag enum with multiple flags set. If the flag enum instance has no flags set or an underlying numeric value of 0 and no specific member defined for 0, the function returns the empty string ("").

See also

DECIMAL function, INTEGER function, ToString( ) method (Object)

SUBSCRIBE statement

Creates a subscription to an ABL named event.

Note: ABL named events are completely different from the key function, mouse, widget, and direct manipulation events described in the Handle-based Object Events Reference. They are also different from the class events described in the Class Events Reference.

Syntax

```
SUBSCRIBE [ PROCEDURE subscriber-handle ] [ TO ] event-name
  { IN publisher-handle | ANYWHERE }
  [ RUN-PROCEDURE local-internal-procedure ] [ NO-ERROR ]
```

`PROCEDURE` *subscriber-handle*

A procedure or handle representing the subscriber.

The `PROCEDURE` option lets one procedure create a subscription on behalf of another. For example, if you want procedure A to create a subscription on behalf of procedure B, set *subscriber-handle* to the procedure handle of B.

If the `PROCEDURE` option does not appear, the AVM creates a subscription on behalf of `THIS-PROCEDURE`, the procedure that contains the `SUBSCRIBE` statement.

`TO` *event-name*

A quoted string or a character expression representing the name of the event.

`IN` *publisher-handle*

Subscribes to the named events published by *publisher-handle*.

If *publisher-handle* is not a valid procedure or widget handle at the time the `SUBSCRIBE` statement executes, the AVM reports a run-time error unless you specify the `NO-ERROR` option.

`ANYWHERE`

Subscribes to named events published within the ABL session, regardless of the publisher.

RUN-PROCEDURE *local-internal-procedure*

A quoted string or character expression representing the name of an internal procedure that resides within the subscribing program. The AVM runs *local-internal-procedure* when the named event occurs.

If the RUN-PROCEDURE option does not appear, when the named event occurs, the AVM runs an internal procedure with the same name as the named event.

Note: The RUN-PROCEDURE option lets you create a subscription when the event name and the procedure name do not match, or when you must subscribe to two different events that have the same name.

When the named event occurs, the AVM RUNs each subscriber's local internal procedure, passing the parameters, if any, The order in which the AVM notifies subscribers is undefined. The AVM always performs this RUN with an implicit NO-ERROR, and logs errors to the ERROR-STATUS system handle.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

For an example, see the reference entry for the PUBLISH statement.

Notes

- Within the local internal procedure, you can get a handle to the publisher of the named event by using the SOURCE-PROCEDURE system handle. For more information on the SOURCE-PROCEDURE system handle, see the reference entry.
- If the AVM detects a redundant SUBSCRIBE statement—that is, a SUBSCRIBE statement with the same event name, and either the same publisher handle or the same ANYWHERE option—the AVM does not report an error.
- If *event-name* is a string containing spaces or is otherwise not a standard ABL name, use one of the following techniques:
  - Use the RUN-PROCEDURE option to assign the local internal procedure a more conventional name.
  - When you define *local-internal-procedure*, put its name in quotes, as in the following example:

```
PROCEDURE "spaced event":
```

See also

NO-ERROR option, PUBLISH statement, PUBLISHED-EVENTS attribute, Subscribe( ) event method, UNSUBSCRIBE statement

SUBSTITUTE function

Returns a character string that is made up of a base string plus the substitution of arguments in the string. It allows you to use a single string in place of concatenated strings. The function is designed to simplify the task of translating an application from one language to another and is similar to the `sprintf` function in the C programming language. The data type returned reflects the data type used in its input parameter. For example, if the input parameter is a LONGCHAR, then the function returns a LONGCHAR; if the input parameter is a CHARACTER, then the function returns a CHARACTER.

Syntax

```
SUBSTITUTE ( base-string [ , arg ] ... )
```

base-string

A CHARACTER or LONGCHAR variable optionally containing substitution parameters of the form &*n*, where *n* is an integer between 1 and 9, inclusive.

arg

A constant, field name, variable, or expression that results in a CHARACTER or LONGCHAR value. These argument values replace substitution parameters in *base-string*.

Example

These statements display the same message:

```
MESSAGE SUBSTITUTE("There were &1 records in &2 tables",
  rec-count, table-count).
```

```
MESSAGE "There were" rec-count "records in"
  table-count "tables".
```

You can alter the position of the substitution parameters, as in this statement:

```
SUBSTITUTE("&2 comes before &1", "Friday", "Monday").
```

Notes

- The SUBSTITUTE function is double-byte enabled. The specified *base-string* and *arg* values can contain double-byte characters.
- To include an ampersand character in *base-string*, enter two ampersands (&&).
- The character following the ampersand character must be a digit, or the AVM returns a run-time error.
- To display the result of the SUBSTITUTE function in a frame, you must specify FORMAT or accept the default format of X(8).

- If you use a substitution parameter in *base-string* but do not specify a corresponding argument, the AVM replaces the substitution parameter with an empty string.
- The SUBSTITUTE function converts Unknown value (?) parameters into empty strings.
- If the length of the resulting substitution exceeds the maximum length for the data type of the original string, an error occurs.
- Any substitution parameter can appear multiple times in *base string*. For example:

```
phrase = "finish on time".
DISPLAY SUBSTITUTE("When I say &1, I mean &1!", phrase) FORMAT "X(70)".
```

The previous code fragment displays the following line:

```
When I say finish on time, I mean finish on time!
```

See also

REPLACE function

SUBSTRING function

Extracts a portion of a character string from a field or variable.

Syntax

```
SUBSTRING ( source , position [ , length [ , type ] ] )
```

source

A CHARACTER or LONGCHAR expression from which you want to extract characters or bytes.

position

An integer expression that indicates the position of the first character you want to extract from *source*.

length

An integer expression that indicates the number of characters you want to extract from *source*. If you do not use the *length* argument or specify -1 as the length, SUBSTRING uses the remainder of the string from the specified *position*.

type

A CHARACTER expression that directs ABL to interpret the specified *position* and *length* values as character units, bytes, or columns. A double-byte character registers as one character unit. By default, ABL interprets the specified *position* and *length* values as character units.

There are four valid types: "CHARACTER," "FIXED," "COLUMN," and "RAW." The expression "CHARACTER" specifies character units. The expression "FIXED" specifies that *position* is in character units and the length is in bytes, but directs SUBSTRING to yield only whole characters. That is, if the last byte or bytes represent part of, but not all of, a multi-byte character, these bytes are excluded. The expression "COLUMN" specifies display or print character-columns. The expression "RAW" specifies bytes. If you specify the *type* as a constant expression, ABL validates the type specification at compile time. If you specify the *type* as a non-constant expression, the AVM validates the type specification at run time.

Note: If *source* is a LONGCHAR expression, "CHARACTER" is the only valid type and the default type.

Example

The `r-substr.p` procedure uses the SUBSTRING function to create invoice numbers. You supply a starting invoice number. The first SUBSTRING function produces the first two characters of today's date; the second SUBSTRING function produces the last two characters of today's date. The procedure concatenates these four characters to a hyphen and the number you entered to produce an invoice number.

r-substr.p

```
DEFINE VARIABLE inv-num AS CHARACTER NO-UNDO FORMAT "x(11)"
  LABEL "Invoice Number".
DEFINE VARIABLE snum    AS INTEGER   NO-UNDO FORMAT "9999"
  LABEL "  Starting Order Number".
DEFINE VARIABLE enum    LIKE snum    NO-UNDO
  LABEL "    Ending Order Number".
DEFINE VARIABLE num     LIKE snum    NO-UNDO
  LABEL "Starting Invoice Number".

UPDATE "      Creating Invoices"
  SKIP(2) snum SKIP(1) enum SKIP(2) num SKIP(2)
  WITH SIDE-LABELS CENTERED NO-BOX.

FOR EACH Order WHERE Order.OrderNum >= snum AND Order.OrderNum <= enum:
  inv-num = SUBSTRING(STRING(TODAY),1,2,"CHARACTER") +
            SUBSTRING(STRING(TODAY),7,2,"CHARACTER") + " - " +
            STRING(num,"9999").
  DISPLAY Order.OrderNum inv-num WITH CENTERED.
  /* Do creation and printing of invoice here */

  num = num + 1.
END.
```

See also

OVERLAY statement, SUBSTRING statement

SUBSTRING statement

Inserts content from a specified expression into a field or variable, optionally replacing existing characters, bytes, or columns.

Syntax

```
SUBSTRING ( target , position [ , length [ , type ] ] ) = expression
```

target

A field or variable of type CHARACTER or LONGCHAR in which you want to store the specified *expression*.

position

An integer expression that indicates the position in the *target* where you want to start storing the *expression*. If the *position* is longer than the *target*, ABL pads the *target* with blanks to equal the length of the *position*.

length

An integer expression that indicates the number of positions you want to replace in the *target*. If you specify a length of 0, the entire *expression* is inserted at the *position* and everything else moves to the right. If you do not use the *length* argument or specify -1 as the length, SUBSTRING puts the entire *expression* into the *target*, replacing an equal amount of the *target*.

type

A character expression that directs ABL to interpret the specified *position* and *length* values as character units, bytes, or columns. A double-byte character registers as one character unit. By default, ABL interprets the specified *position* and *length* values as character units.

There are three valid *type* values: "CHARACTER", "RAW", and "COLUMN". The value "CHARACTER" specifies character units. The value "RAW" specifies bytes. The value "COLUMN" specifies display or print character-columns. If you specify the *type* as a constant expression, ABL validates the *type* specification at compile time. If you specify the *type* as a non-constant expression, the AVM validates the *type* specification at run time.

Note: If the *target* is a LONGCHAR expression, "CHARACTER" is the only valid *type* value.

expression

A constant, field name, variable name, or expression of CHARACTER or LONGCHAR data type that results in a character string whose value you want to insert in the *target*. ABL does not pad or truncate *expression*.

Note: The += Addition assignment operator can be used with the SUBSTRING statement.

Examples

The following procedure uses the SUBSTRING statement to replace a segment of text with the expression in the SUBSTRING statement XXXXXXXXX. The procedure first displays the text you can work with in the Original Text frame. Then the procedure prompts you for the start position of the replacement and the length of the replacement. Under the WORD heading, you see the revised text.

r-sub.p

```
DEFINE VARIABLE rtext AS CHARACTER NO-UNDO FORMAT "x(50)".
DEFINE VARIABLE orig  AS CHARACTER NO-UNDO FORMAT "x(31)".
DEFINE VARIABLE strt  AS INTEGER   NO-UNDO FORMAT ">9".
DEFINE VARIABLE leng  AS INTEGER   NO-UNDO FORMAT ">9".

orig = "Now is the time to use OpenEdge".
DISPLAY orig WITH CENTERED TITLE "Original Text" NO-LABEL.
REPEAT:
  rtext = orig.
  UPDATE strt LABEL "START" leng LABEL "LENGTH".
  SUBSTRING(rtext, strt, leng, "CHARACTER") = "XXXXXXXXX".
  DISPLAY rtext LABEL "WORD" WITH CENTERED.
END.
```

The SUBSTRING and OVERLAY statements use the *length* option differently. For both, the *length* indicates how much of the *target* to replace. However, SUBSTRING always inserts the full *expression* and never pads the *expression* to match the *length*. By contrast, the *length* in OVERLAY determines how much ABL adds to the *target*, even if ABL must truncate the *expression* or pad it with spaces. The following procedure illustrates the differences between these two statements.

r-sub-over.p

```
/* This procedure illustrates the differences between the SUBSTRING and
   OVERLAY statements. */
DEFINE VARIABLE cOriginal  AS CHARACTER NO-UNDO INITIAL "OpenEdge".
DEFINE VARIABLE cSubstring AS CHARACTER NO-UNDO.
DEFINE VARIABLE cOverlay   AS CHARACTER NO-UNDO.
DEFINE VARIABLE cResults   AS CHARACTER NO-UNDO.

/* Default behavior without optional LENGTH. */
ASSIGN
 cSubstring              = cOriginal
 SUBSTRING(cSubstring,2) = "***"
 cOverlay                = cOriginal
 OVERLAY(cOverlay,2)     = "***"
 cResults                = "target = ~"OpenEdge~". ~n~n"
  + "If you do not supply a length, SUBSTRING and OVERLAY default as follows:
  ~n~n" + "SUBSTRING(target,2) = ~"***~"  yields:  " + cSubstring + ". ~n"
  + "OVERLAY(target,2)     = ~"***~"  yields:  " + cOverlay + ".".

/* Behavior with zero LENGTH. */
ASSIGN
 cSubstring                = cOriginal
 SUBSTRING(cSubstring,2,0) = "***"
 cOverlay                  = cOriginal
 OVERLAY(cOverlay,2,0)     = "***"
 cResults                  = cResults + "~n~n"
  + "For a zero length, SUBSTRING and OVERLAY behave as follows:  ~n~n"
  + "SUBSTRING(target,2,0) = ~"***~"  yields:  " + cSubstring + ". ~n"
  + "OVERLAY(target,2,0)     = ~"***~"  yields:  " + cOverlay + ".".

/* Behavior with LENGTH < replacement. */
ASSIGN
 cSubstring                = cOriginal
 SUBSTRING(cSubstring,2,1) = "***"
 cOverlay                  = cOriginal
 OVERLAY(cOverlay,2,1)     = "***"
 cResults                  = cResults + "~n~n"
  + "For a length shorter than the replacement, SUBSTRING and OVERLAY behave
  as follows: ~n~n" + "SUBSTRING(target,2,1) = ~"***~"  yields:  "
  + cSubstring + ". ~n" + "OVERLAY(target,2,1)     = ~"***~"  yields:  "
  + cOverlay + ".".

/* Behavior with LENGTH = replacement. */
```

```
ASSIGN
 cSubstring                = cOriginal
 SUBSTRING(cSubstring,2,3) = "***"
 cOverlay                  = cOriginal
 OVERLAY(cOverlay,2,3)     = "***"
 cResults                  = cResults + "~n~n"
  + "For a length equal to the replacement, SUBSTRING and OVERLAY behave as
  follows:  ~n~n" + "SUBSTRING(target,2,3) = ~"***~"  yields:  "
  + cSubstring + ". ~n" + "OVERLAY(target,2,3)     = ~"***~"  yields:  "
  + cOverlay + ".".

/* Behavior with LENGTH > replacement. */
ASSIGN
 cSubstring                = cOriginal
 SUBSTRING(cSubstring,2,6) = "***"
 cOverlay                  = cOriginal
 OVERLAY(cOverlay,2,6)     = "***"
 cResults                  = cResults + "~n~n"
  + "For a length greater than the repalcement, SUBSTRING and OVERLAY behave
  as follows:  ~n~n" + "SUBSTRING(target,2,6) = ~"***~"  yields:  "
  + cSubstring + ". ~n" + "OVERLAY(target,2,6)     = ~"***~"  yields:  "
  + cOverlay + ".".

MESSAGE cResults VIEW-AS ALERT-BOX.
```

Notes

- Do not split double-byte characters. This statement allows you to replace either the lead- or trail-byte of the target string when you specify "RAW" for the *type* parameter.

See also

OVERLAY statement, SUBSTRING function

SUPER function

Runs the super procedure version of the current user-defined function.

This language element must appear within a user-defined function, but can appear anywhere within the user-defined function. If this language element does not appear within a user-defined function, the compiler reports an error.

Syntax

```
SUPER [ ( parameter [ , parameter ] ... ) ]
```

parameter

A parameter of the super version of the current user-defined function. These parameters must have the same signature (number of parameters, and type and mode of each) as the parameters of the current user-defined function. You can, however, adjust a parameter's value.

For the *parameter* syntax, see the Parameter definition syntax reference entry.

If a user-defined function cannot be located in any super procedure, the AVM generates the following error message:

```
SUPER version of user-defined function name invoked but could not be found
```

Errors are stored in the ERROR-STATUS handle when NO-ERROR is specified.

Example

For an example of the SUPER function, see the RUN SUPER statement reference entry.

Notes

- To run the super version of an internal procedure, use the RUN SUPER statement.
- For the rules that ABL uses to find the super version of the current user-defined function, see the ADD-SUPER-PROCEDURE( ) method reference entry.

See also

ADD-SUPER-PROCEDURE( ) method, REMOVE-SUPER-PROCEDURE( ) method, RUN SUPER statement, SOURCE-PROCEDURE system handle, SUPER-PROCEDURES attribute, TARGET-PROCEDURE system handle

SUPER statement

Invokes a constructor for the immediate super class as the first statement in a constructor of the defining class.

Syntax

```
SUPER ( [ parameter [ , parameter ] ... ] )
```

([*parameter* [, *parameter*] ...])

Specifies zero or more parameters passed to a PACKAGE-PRIVATE, PROTECTED, PACKAGE-PROTECTED, or PUBLIC constructor that is defined for the super class. You must provide the parameters identified by the specified constructor, matched with respect to number, data type, and mode. To invoke a constructor that is overloaded in the class, you must specify sufficient information for each *parameter* to disambiguate it from all other constructors in the class. Otherwise, ABL raises an error identifying the ambiguity.

For information on the parameter passing syntax and disambiguating overloaded constructors, see the Parameter passing syntax reference entry.

Notes

- You must make sure that at least one constructor is called in each level of the class hierarchy. If there is no constructor defined in the immediate super class or there is a constructor defined that does not take parameters, you do not need to explicitly invoke it, although you can. By default, ABL will call it. However if you want to invoke a super class constructor with parameters, you must explicitly invoke it. This is normally

done using SUPER as the first executable statement of the constructor. Alternatively you can call another local constructor using the THIS-OBJECT statement and then use SUPER from there.

See also

CONSTRUCTOR statement, Parameter passing syntax, THIS-OBJECT statement, SUPER system reference

SUPER system reference

A system reference that lets a subclass call the PUBLIC, PACKAGE-PROTECTED, PROTECTED, and PACKAGE-PRIVATE instance methods of its super class in the inherited class hierarchy. If the specified method definition is not found in the subclass's immediate super class, ABL repeatedly looks to the next super class in the inherited class hierarchy until it finds the definition.

Syntax

```
SUPER:method-name ( [ parameter [ , parameter ] ... ] ) [ NO-ERROR ]
```

method-name

Specifies the name of an instance method defined in a super class. The method definition cannot be abstract or ABL raises a compiler error.

([*parameter* [, *parameter*] ...])

Specifies zero or more parameters passed to a PACKAGE-PRIVATE, PROTECTED, PACKAGE-PROTECTED, or PUBLIC method that is defined for the super class. You must provide the parameters identified by the specified method, matched with respect to number, data type, and mode. To invoke a method that is overloaded in the class, you must specify sufficient information for each *parameter* in order to disambiguate it from all the other methods that it overloads. Otherwise, ABL raises an error identifying the ambiguity.

For information on the parameter passing syntax and disambiguating overloaded methods, see the Parameter passing syntax reference entry.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Notes

- You typically use the SUPER system reference within a method of a class defined somewhere in the hierarchy to invoke a method defined in a super class that was overridden in a subclass. For more information about using the SUPER system reference, see *Develop Object-oriented ABL Applications*.
- You **cannot** use the SUPER system reference to call a static method.

See also

Class-based method call, NO-ERROR option, Parameter passing syntax, SUPER statement

SYSTEM-DIALOG COLOR statement
(Windows only; Graphical interfaces only)

Displays a dialog box that lets the user choose and associate a system color with the specified dynamic color number. The SYSTEM-DIALOG COLOR statement provides a dialog box appropriate to the graphical environment in which it runs.

Note: Does not apply to SpeedScript programming.

Syntax

```
SYSTEM-DIALOG COLOR color-number
   [ UPDATE logical-variable ]
   [ IN WINDOW window ]
```

color-number

An integer expression that evaluates to an ABL color number from 0 to 255, inclusive, that is defined as dynamic through the SET-DYNAMIC method of the COLOR-TABLE handle. The color dialog associates the ABL color specified by *color-number* with the system color value the user selects in the dialog box. The user chooses the OK button to confirm the choice. The user can close the dialog box without changing the color by choosing the Cancel button.

UPDATE *logical-variable*

Specifies a logical variable to return the status of the user's color dialog interaction. If the user chooses the OK button, the dialog sets *logical-variable* to TRUE. If the user chooses the Cancel button, the dialog sets *logical-variable* to FALSE.

IN WINDOW *window*

Specifies the window where the dialog box is displayed. The value *window* must be the handle of a window.

Example

The following procedure displays a dialog box that allows the user to assign new foreground and background colors to the dialog box. A radio set in the dialog box lists selections for foreground and background that correspond to the numbers nine and eight, respectively. Choosing the OK button opens a color dialog box to assign a new system color to the selected color number. Note that the UPDATE option is not used to return a termination status because the dialog does not require the user to select a new color; it only provides the option. The procedure terminates when the user chooses the Cancel button in the radio selection dialog box.

r-coldlg.p

```
DEFINE VARIABLE front-color AS INTEGER NO-UNDO INITIAL 9.
DEFINE VARIABLE back-color  AS INTEGER NO-UNDO INITIAL 8.
DEFINE VARIABLE curr-color  AS INTEGER NO-UNDO INITIAL 9
  VIEW-AS RADIO-SET RADIO-BUTTONS "Foreground", 9, "Background", 8.

DEFINE BUTTON ok-button     LABEL "OK".
DEFINE BUTTON cancel-button LABEL "Cancel" AUTO-ENDKEY.

FORM
  SKIP(0.5) SPACE(0.5)
  curr-color SPACE(2) ok-button SPACE(2) cancel-button
  SPACE(0.5) SKIP(0.5)
  WITH FRAME color-frame NO-LABELS TITLE "Choose frame colors ..."
    FGCOLOR front-color BGCOLOR back-color VIEW-AS DIALOG-BOX.

ON CHOOSE OF ok-button IN FRAME color-frame DO:
  ASSIGN curr-color.
  IF NOT COLOR-TABLE:GET-DYNAMIC(curr-color) AND
     NOT COLOR-TABLE:SET-DYNAMIC(curr-color,TRUE) THEN
    MESSAGE "Color must be DYNAMIC to edit.".
  ELSE
    SYSTEM-DIALOG COLOR curr-color.
END.

UPDATE curr-color ok-button cancel-button WITH FRAME color-frame.
```

Note that the trigger for the ok-button must assign the curr-color variable to obtain the latest value selected for the radio set. The GET-DYNAMIC and SET-DYNAMIC methods are used to ensure that the color is dynamic before modifying it.

Notes

- For more information on defining dynamic colors, see *Manage ABL Applications*.
- Use the *color-number* in a COLOR phrase to assign the selected color to a widget.

See also

COLOR phrase, COLOR-TABLE system handle

SYSTEM-DIALOG FONT statement
(Windows only; Graphical interfaces only)

Displays a dialog box that allows the user to select and associate a system font with the specified font number. The SYSTEM-DIALOG FONT statement provides a dialog box appropriate to the graphical environment in which it runs.

Note: Does not apply to SpeedScript programming.

Syntax

```
SYSTEM-DIALOG FONT font-number
   [ ANSI-ONLY ]
   [ FIXED-ONLY ]
   [ MAX-SIZE point-size ]
   [ MIN-SIZE point-size ]
   [ UPDATE logical-variable ]
   [ IN WINDOW window ]
```

font-number

An integer expression that returns an ABL font number (0 to 255), inclusive, which is defined in the setup file for your environment. The font dialog associates the ABL font specified by *font-number* with the system font the user selects in the dialog. The user confirms the selection and completes the dialog by choosing the OK button. The user interrupts the dialog without changing the font by choosing the Cancel button.

ANSI-ONLY

Allows the font dialog to provide only fonts that contain character representations and that do not include graphic symbols.

FIXED-ONLY

Allows the font dialog to provide only mono-spaced fonts.

MAX-SIZE *point-size*

Has no effect; supported only for backward compatibility.

MIN-SIZE *point-size*

Has no effect; supported only for backward compatibility.

UPDATE *logical-variable*

Specifies a logical variable to return the status of the user's font dialog interaction. If the user clicks on the OK button, the dialog sets *logical-variable* to TRUE. If the user chooses on the Cancel button, the dialog sets *logical-variable* to FALSE.

IN WINDOW *window*

Specifies the window from which the dialog box is displayed. The value *window* must be the handle of a window.

Example

The following procedure displays a dialog box that allows the user to change the font of either its radio set or its buttons. The radio set lists a font number for each selection: font 1 for the radio set and font 2 for the buttons. Choosing the OK button opens a font dialog to assign a new system font to the font number selected in the radio set. Note that the UPDATE option is not used to return a termination status because the dialog does not require the user to select a new font; it only provides the option. The procedure terminates when the user chooses the Cancel button.

r-fntdlg.p

```
DEFINE VARIABLE Font1      AS INTEGER NO-UNDO INITIAL 1.
DEFINE VARIABLE Font2      AS INTEGER NO-UNDO INITIAL 2.
DEFINE VARIABLE FontSelect AS INTEGER NO-UNDO INITIAL 1
  VIEW-AS RADIO-SET RADIO-BUTTONS "Font 1", 1, "Font 2", 2 FONT Font1.

DEFINE BUTTON bOK     LABEL "OK" FONT Font2.
DEFINE BUTTON bCANCEL LABEL "Cancel" FONT Font2 AUTO-ENDKEY.

FORM
  SKIP(0.5) SPACE(0.5)
  FontSelect SPACE(2) bOK SPACE(2) bCANCEL
  SPACE(0.5) SKIP(0.5)
  WITH FRAME fFont TITLE "Choose frame fonts ..." VIEW-AS DIALOG-BOX.

ON CHOOSE OF bOK IN FRAME fFont DO:
  IF INTEGER(FontSelect:SCREEN-VALUE IN FRAME fFont) = Font1 THEN
    SYSTEM-DIALOG FONT Font1.
  ELSE
    SYSTEM-DIALOG FONT Font2.
END.

UPDATE FontSelect bOK bCANCEL WITH FRAME fFont.
```

Note: The CHOOSE OF OK event trigger must reference the SCREEN-VALUE attribute of the FontSelect variable to obtain the latest value selected for its radio set. This is because the UPDATE statement has not yet completed during the event, and has not updated the FontSelect record buffer from the frame buffer. The initial value of FontSelect is its value in the record buffer immediately before the UPDATE statement executes.

Notes

- For more information on defining font numbers, see *Manage ABL Applications*.
- Use the *font-number* with the FONT option to assign the selected font to a widget.

SYSTEM-DIALOG GET-DIR statement
(Windows only)

Displays a dialog box that allows the user to enter a directory name that is assigned to a character variable. The SYSTEM-DIALOG GET-DIR statement provides a dialog box appropriate to the environment in which it runs.

Note: Does not apply to SpeedScript programming.

Syntax

```
SYSTEM-DIALOG GET-DIR character-field
  [ INITIAL-DIR directory-string ]
  [ RETURN-TO-START-DIR ]
  [ TITLE title-string ]
```

character-field

The character field or variable that contains the directory name the user enters. The user can enter the directory name by typing it or selecting it from a list of directories in the common dialog directory. The user confirms the entry and completes the dialog by choosing the OK button. The user can interrupt the dialog without any selection by choosing the Cancel button. The directory name can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

INITIAL-DIR *directory-string*

Sets the starting directory for this invocation of SYSTEM-DIALOG GET-DIR to the pathname specified in *directory-string* before starting the dialog. The *directory-string* is a character expression that must evaluate to a valid pathname in your environment. The default starting directory is either the current working directory or the directory left from the last invocation of SYSTEM-DIALOG GET-DIR. The directory name can contain Unicode characters.

RETURN-TO-START-DIR

This option resets the current directory to the starting directory when the common dialog ends. This is the directory specified by the INITIAL-DIR option or the default starting directory.

If you do not specify this option, the directory remains set at the last directory referenced by the user. This directory becomes the default initial directory for subsequent invocations of SYSTEM-DIALOG GET-DIR. This option also has no effect on subsequent invocations that specify the INITIAL-DIR option.

TITLE *title-string*

Specifies a title for the dialog box. The value *title-string* can be any character expression. If you do not specify a title, the dialog uses the system default for your environment.

Notes

- The default common dialog directory for the initial invocation of SYSTEM-DIALOG GET-DIR is the current working directory. You can specify a different starting common dialog directory with the INITIAL-DIR option and the user can change the common dialog directory by referencing a different directory in the common dialog.
- The Windows common dialog never searches the PROPATH, and always returns the full pathname of the entered relative pathname appended to the current common dialog directory.

SYSTEM-DIALOG GET-FILE statement
(Windows only)

Displays a dialog box that allows the user to enter a filename that is assigned to a character variable. The SYSTEM-DIALOG GET-FILE statement provides a dialog box appropriate to the environment in which it runs.

Note: Does not apply to SpeedScript programming.

Syntax

```
SYSTEM-DIALOG GET-FILE character-field
  [ FILTERS name filespec
      [ , name filespec ] ...
      [ INITIAL-FILTER filter-num ]
  ]
  [ ASK-OVERWRITE ]
  [ CREATE-TEST-FILE ]
  [ DEFAULT-EXTENSION extension-string ]
  [ INITIAL-DIR directory-string ]
  [ MUST-EXIST ]
  [ RETURN-TO-START-DIR ]
  [ SAVE-AS ]
  [ TITLE title-string]
  [ USE-FILENAME ]
  [ UPDATE logical-variable ]
  [ IN WINDOW window ]
```

character-field

The character field or variable that contains the filename the user enters. The user can enter the filename by typing it or selecting it from a list of files in the common dialog directory. The user confirms the entry and completes the dialog by choosing the OK button. The user can interrupt the dialog without any selection by choosing the Cancel button.

You can also use *character-field* to pass a default filename entry to the dialog. See the USE-FILENAME option for more information.

The filename can contain Unicode characters.

FILTERS *namefilespec*

Defines one or more filters for the filename dialog. Each filter selects a subset of the available files in the common dialog directory to build the dialog file selection-list. A filter consists of two parts: a label (*name*) and file specification (*filespec*).

The *name* is a character expression used as a label for your filter. Windows uses the label to identify the filter in a filter selection-list. The user can select the label to view the list of files selected by the filter.

The *filespec* is a character expression that evaluates to a file specification string. This string can consist of any wild cards or regular expressions used to generate valid file specifications in your environment. In Windows, *filespec* can also consist of multiple file specifications, separating each one with a comma, for example: *.p,*.i,*.r.

If you do not specify any filters, the dialog builds the selection-list with all files in the directory.

INITIAL-FILTER *filter-num*

Specifies the initial filter list defined by the FILTERS option, where *filter-num* is an integer expression that evaluates to the position of the filter in the list, starting from 1.

If you do not specify the INITIAL-FILTER option, the dialog uses the first filter in the list as the initial filter.

`ASK-OVERWRITE`

Causes a the dialog to prompt for confirmation if the user enters the name of a file that already exists. By default, the dialog does not prompt for confirmation if the user enters an existing filename. In Windows, this option is ignored unless SAVE-AS is also specified.

`CREATE-TEST-FILE`

Causes the filename dialog to create a temporary file before it completes in order to verify that the user has write access to the directory path specified for the filename entry. If the dialog cannot write the file, it displays an error message and prompts for another filename entry. The dialog does not complete until the user enters a filename associated with a writable directory or chooses the Cancel button to interrupt the dialog. After successful completion, the dialog deletes the temporary file.

This option is especially appropriate with the SAVE-AS option to verify the ability to save a file.

`DEFAULT-EXTENSION` *`extension-string`*

Specifies a default extension (or suffix) to be appended to the user's filename entry after completing the filename dialog, where *`extension-string`* is a character expression that evaluates to a valid file extension in your environment, including all required punctuation. In Windows, the extension must start with a period.

The Windows dialog appends the specified extension to the user's filename entry only if the entry does not already contain an extension.

`INITIAL-DIR` *`directory-string`*

Sets the starting directory for this invocation of SYSTEM-DIALOG GET-FILE to the pathname specified in *`directory-string`* before starting the dialog. The *`directory-string`* is a character expression that must evaluate to a valid pathname in your environment. The default starting directory is either the current working directory or the directory left from the last invocation of SYSTEM-DIALOG GET-FILE.

`MUST-EXIST`

Requires that the user's filename entry, complete with any specified default extension, must exist in the directory specified for the filename entry before the dialog completes. If it does not exist, the dialog displays an error message and prompts for another filename entry. The dialog does not complete until the user enters the name of an existing file or chooses the Cancel button to interrupt the dialog.

`RETURN-TO-START-DIR`

This option resets the current directory to the starting directory when the common dialog ends. This is the directory specified by the INITIAL-DIR option or the default starting directory.

If you do not specify this option, the directory remains set at the last directory referenced by the user. This directory becomes the default initial directory for subsequent invocations of SYSTEM-DIALOG GET-FILE. This option also has no effect on subsequent invocations that specify the INITIAL-DIR option.

SAVE-AS

Causes the dialog box to become a Save As dialog box. For a Save As dialog box, the default box title is "Save As". You can use the ASK-OVERWRITE option with SAVE-AS to get confirmation before accepting an existing file from the dialog.

TITLE *title-string*

Specifies a title for the dialog box. The value *title-string* can be any character expression. If you do not specify a title, the dialog uses the system default for your environment.

USE-FILENAME

Specifies the contents of *character-field* as the default filename entry for the dialog. The contents can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

During the dialog, the user can accept the default entry or override it by entering or selecting another filename.

UPDATE *logical-variable*

Specifies a logical variable to return the status of the user's filename dialog interaction. If the user chooses the OK button, the dialog sets *logical-variable* to TRUE. If the user chooses the Cancel button, the dialog sets *logical-variable* to FALSE.

IN WINDOW *window*

Specifies the window from which the dialog box is displayed. The value *window* must be the handle of a window.

Example

The following example uses the filename dialog box to run procedures. It allows the user to select and run procedure files until they choose the Cancel button.

r-fildlg.p

```
DEFINE VARIABLE procname  AS CHARACTER NO-UNDO.
DEFINE VARIABLE OKpressed AS LOGICAL   NO-UNDO INITIAL TRUE.

Main:
REPEAT:
  SYSTEM-DIALOG GET-FILE procname
    TITLE   "Choose Procedure to Run ..."
    FILTERS "Source Files (*.p)"   "*.p",
            "R-code Files (*.r)"   "*.r"
    MUST-EXIST
    USE-FILENAME
    UPDATE OKpressed.

  IF OKpressed = TRUE THEN
    RUN VALUE(procname).
  ELSE LEAVE Main.
END.
```

Notes

- The default common dialog directory for the initial invocation of SYSTEM-DIALOG GET-FILE is the current working directory. You can specify a different starting common dialog directory with the INITIAL-DIR option and the user can change the common dialog directory by referencing a different directory in the common dialog.
- The Windows common dialog never searches the PROPATH, and always returns the full pathname of the entered relative pathname appended to the current common dialog directory.

SYSTEM-DIALOG PRINTER-SETUP statement
(Windows only)

Displays the Windows Print dialog box and lets the user set the default print context for subsequent print jobs in Windows.

Note: Does not apply to SpeedScript programming.

Syntax

```
SYSTEM-DIALOG PRINTER-SETUP
   [ NUM-COPIES expression ]
   [ LANDSCAPE | PORTRAIT ]
   [ UPDATE status ]
   [ IN WINDOW window ]
```

NUM-COPIES *expression*

Specifies the initial value of the Copies field in the Print dialog box. The value *expression* must evaluate to an integer expression. The user can change this value within the dialog box. This option is supported only with printer drivers that support multi-copy printing. Otherwise, the Copies field is disabled.

LANDSCAPE

Specifies the initial value of the Orientation field in the Properties dialog box as landscape. The user can change this value within the dialog box. The Properties dialog box is accessible from the Print dialog box. This option is supported only with printer drivers that support landscape page orientation.

PORTRAIT

Specifies the initial value of the Orientation field in the Properties dialog box as portrait. The user can change this value within the dialog box. The Properties dialog box is accessible from the Print dialog box. This option is supported only with printer drivers that support portrait page orientation.

UPDATE *status*

Specifies a logical variable to return the status of the user's dialog interaction. If the user chooses the OK button, the dialog sets *status* to TRUE. If the user chooses the Cancel button, the dialog sets *status* to FALSE.

`IN WINDOW` *`window`*

Specifies the window from which the Print dialog box is displayed. The value *`window`* must be the handle of a window.

Example

This example presents a dialog box that allows you to set up and print information from the Sports2020 database. When you choose the Printer Setup button, it displays the Windows Print dialog box. Using the latest settings, you can then print a list of customer names from the Sports2020 database in alphabetical order by choosing the Print Customer Names button.

r-prtdlg.p

```
DEFINE BUTTON bprintset   LABEL "Printer Setup".
DEFINE BUTTON bprintnames LABEL "Print Customer Names".
DEFINE BUTTON bcancel     LABEL "Cancel".

DEFINE FRAME PrintFrame
  bprintset bprintnames bcancel
  WITH TITLE "Quick Printer" VIEW-AS DIALOG-BOX.

ON CHOOSE OF bprintset DO:
  SYSTEM-DIALOG PRINTER-SETUP.
END.

ON CHOOSE OF bprintnames DO:
  OUTPUT TO PRINTER.
  FOR EACH Customer NO-LOCK BY Customer.Name:
    DISPLAY Customer.Name WITH STREAM-IO.
  END.
  OUTPUT CLOSE.
END.

ENABLE ALL WITH FRAME PrintFrame.
WAIT-FOR CHOOSE OF bcancel IN FRAME PrintFrame.
```

Notes

- The default print context is the set of values that defines the default printer and setup for that printer in Windows. If there is no default print context, the AVM uses the printer control settings from the current environment.
- Use the PRINTER-NAME attribute of the SESSION system handle to set the printer name in the default print context without user intervention.
- By default, the OUTPUT TO PRINTER statement prints jobs based on the default print context. However, you can use the OUTPUT TO PRINTER statement with its various options to override the default print context for a specific print job.

See also

OUTPUT TO statement, SESSION system handle

SYSTEM-HELP statement
(Windows only)

The SYSTEM-HELP statement calls the Microsoft Windows Help engine to display Windows Help topics, and the HTML Help engine to display HTML Help topics.

Note: Does not apply to SpeedScript programming.

Syntax

```
SYSTEM-HELP file-string
  [ WINDOW-NAME window-name ]
  {   CONTENTS
    | CONTEXT int-expr
    | HELP-TOPIC string
    | KEY string
    | ALTERNATE-KEY string
    | POSITION X x Y y WIDTH dx HEIGHT dy
    | POSITION MAXIMIZE
    | QUIT
    | SET-CONTENTS int-expr
    | CONTEXT-POPUP int-expr
    | PARTIAL-KEY string
    | MULTIPLE-KEY char TEXT string
    | COMMAND string
    | FINDER
    | FORCE-FILE
    | HELP
  }
```

file-string

The *file-string* parameter is a character expression that specifies the pathname of a help file. If the file has a `.chm` extension (the extension for compiled Microsoft HTML Help files), the Microsoft HTML Help viewer is launched. If the file has a `.hlp` file extension, the Microsoft Windows Help viewer is launched. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

`WINDOW-NAME` *window-name*

This option is supported for Windows Help (`.hlp` files) only.

The *window-name* parameter is a character expression that evaluates to the primary or secondary window name as defined in the [WINDOWS] section of the help project file. If the window name is omitted, or if "main" is specified, the primary help window is used.

`CONTENTS`

Supported only for backward compatibility.

For HTML Help, this option displays the Microsoft HTML Help viewer with the default topic in the content pane. Use the HELP-TOPIC option to specify the topic to display.

For Windows Help, this option displays the help topic defined as the contents in the [OPTIONS] section of the help project file.

`CONTEXT` *`int-expr`*

Displays the help topic that the context number identifies. You define context numbers in the [MAP] section of the help project file.

The *`int-expr`* parameter is the context number for the help topic.

`HELP-TOPIC` *`string`*

This option is supported for HTML Help (`.chm` files) only.

Displays a help topic in the content pane of the Microsoft HTML Help viewer.

The *`string`* parameter is a character expression that indicates the topic (`.htm`/`.html` file) within the compiled Microsoft HTML Help (`.chm`) file to display.

`KEY` *`string`*

For HTML Help, this option displays the topic matching the string found in the keyword index. Use semicolons in the *`string`* parameter to delimit multiple keywords. If no match is found, Microsoft HTML Help displays the help viewer with the Index tab on top.

For Windows Help, this option displays the help topic matching the string found in the index keyword list. If there is more than one match, it displays the first topic containing the keyword. If there is no match or the string is omitted, a message is displayed indicating that the keyword is invalid. The *`string`* parameter is a character expression that evaluates to a keyword for the desired help topic.

`ALTERNATE-KEY` *`string`*

This option is supported for HTML Help (`.chm` files) only. For Windows Help (`.hlp` files), see the MULTIPLE-KEY option.

Displays a help topic matching the *`string`* found in the alternate keyword (Alink) index.

The *`string`* parameter is a character expression that evaluates to a keyword in the alternate keyword index.

`POSITION X` *`x`* `Y` *`y`* `WIDTH` *`dx`* `HEIGHT` *`dy`*

Positions an existing (already opened) help window as specified.

The *x* parameter is an integer expression that specifies the x coordinate for the help window.

The *y* parameter is an integer expression that specifies the y coordinate for the help window.

The *dx* parameter is an integer expression that specifies the width of the help window.

The *dy* parameter is an integer expression that specifies the height of the help window.

`POSITION MAXIMIZE`

Maximizes an existing (already opened) help window.

`QUIT`

Informs the help application that help is no longer required. If no other applications are using help, the operating system closes the help application.

`SET-CONTENTS` *`int-expr`*

Supported only for backward compatibility. This option is supported for Windows Help (`.hlp` files) only.

Dynamically re-maps the contents help topic from what is defined in the [OPTIONS] section of the help project file. When a CONTENTS call is made, the new contents help topic is displayed.

The *`int-expr`* parameter is the context number for the new contents help topic.

`CONTEXT-POPUP` *`int-expr`*

This option is supported for Windows Help (`.hlp` files) only.

Displays the help topic in a pop-up window that the context number identifies. You define context numbers in the [MAP] section of the help project file. If a non-scrolling region exists in a help topic, only that region displays when you use the CONTEXT-POPUP option to display the topic.

The *`int-expr`* parameter is the context number for the help topic.

`PARTIAL-KEY` *`string`*

This option is supported for Windows Help (`.hlp` files) only.

Displays the help topic matching the string found in the keyword list. In Windows, if there is more than one match, no match, or if the string is omitted, it displays the Help Topics: Window Help Topics dialog box with the Index tab on top.

The *`string`* parameter is a character expression that evaluates to a partial key for the desired help topic.

`MULTIPLE-KEY` *`char`* `TEXT` *`string`*

This option is supported for Windows Help (`.hlp` files) only. For HTML Help, see the ALTERNATE-KEY option.

Displays the help topic matching a keyword from an alternate keyword table.

The *`char`* parameter is a character expression that evaluates to the single character keyword table identifier for the required table.

The *`string`* parameter is a character expression that evaluates to the keyword that is located in the keyword table.

`COMMAND` *`string`*

This option is supported for Windows Help (`.hlp` files) only.

Executes a help macro.

The *`string`* parameter is a character expression that evaluates to the help macro to execute.

`FINDER`

This option is supported for Windows Help (`.hlp` files) only.

Displays the Help Topics: Windows Help Topics dialog box, which contains an Index tab, a Find tab, and optionally a Contents tab, with the most recently used tab displayed on top.

If a Contents tab file (`.CNT` file) is present when you initially call the Help Topics: Windows Help dialog box, then the Content tab displays on top. However, if a `.CNT` file is not present, then the dialog box displays with the Index tab on top; the Contents tab is not available.

FORCE-FILE

This option is supported for Windows Help (.hlp files) only.

Ensures that the correct help file is open and displayed.

HELP

This option is supported for Windows Help (.hlp files) only.

Displays the contents of the ABL Help-on-Help file. In Windows, HELP displays the Help Topics: Windows Help Topics dialog box.

Example

The following example demonstrates several features of the SYSTEM-HELP statement with the Procedure Editor help file (editeng.chm). The user can select a button to demonstrate each of the following SYSTEM-HELP options: CONTEXT, KEY, ALTERNATE-KEY, POSITION, POSITION-MAXIMIZE, and QUIT.

To execute this procedure, first copy the editeng.chm file from DLC\prohelp to your current working directory (by default, C:\OpenEdge\WRK). Then open and run r-syshlpchm.p in the Procedure Editor.

r-syshlpchm.p

```
DEFINE VARIABLE helpfile AS CHARACTER NO-UNDO.

DEFINE BUTTON b_context LABEL "CONTEXT Call".
DEFINE BUTTON b_blank   LABEL "KEY Call-''".
DEFINE BUTTON b_single  LABEL "KEY Call-'Tools'".
DEFINE BUTTON b_full    LABEL "KEY Call- Tools;Menu".
DEFINE BUTTON b_max     LABEL "POSITION MAXIMIZE Call".
DEFINE BUTTON b_pos     LABEL "POSITION Call".
DEFINE BUTTON b_alt     LABEL "ALTERNATE-KEY Call".
DEFINE BUTTON b_quit    LABEL "QUIT Call".

FORM
  SKIP(1) SPACE(1)(1) b_context SPACE(1)
  SKIP(1) SPACE(1) b_blank SPACE(1)
  SKIP(1) SPACE(1) b_single SPACE(1)
  SKIP(1) SPACE(1) b_full SPACE(1)
  SKIP(1) SPACE(1) b_max SPACE(1)
  SKIP(1) SPACE(1) b_pos SPACE(1)
  SKIP(1) SPACE(1) b_alt SPACE(1)
  SKIP(1) SPACE(1) b_quit SPACE(1)
  SKIP(1) WITH FRAME x.
ENABLE ALL WITH FRAME x.

helpfile = "editeng.chm".

/* The CONTEXT call displays the help topic associated with the specified
   context number of a help topic (in this case, 49256, for the Using Editor
   Buffers topic). */
ON CHOOSE OF b_context IN FRAME x DO:
  SYSTEM-HELP helpfile CONTEXT 49256.
END.

/* The KEY call brings up the topic matching the string found in the keyword
   index. If the string parameter is empty or is omitted altogether, the help
   viewer displays with the Index tab on top.*/
ON CHOOSE OF b_blank IN FRAME x DO:
  SYSTEM-HELP helpfile KEY "".
END.

/* In a KEY call where the string parameter does not exactly match an index
   keyword of any help topic, the fill-in at the top of the Index tab is
```

```
   populated with the string that is passed in, and the default help topic
   is displayed. */
ON CHOOSE OF b_single IN FRAME x DO:
  SYSTEM-HELP helpfile KEY "Tools".
END.

/* In a KEY call where the string parameter exactly matches a unique index
   keyword of a help topic, the help engine automatically launches a help
   viewer window and displays the matching topic. Use semicolons to delimit
   multiple keywords.*/
ON CHOOSE OF b_full IN FRAME x DO:
  SYSTEM-HELP helpfile KEY "Tools;Menu".
END.

/* In an ALTERNATE-KEY call works like the KEY call but it uses the alternate
   keyword (Alink) index, if one is provided. */
ON CHOOSE OF b_alt IN FRAME x DO:
  SYSTEM-HELP helpfile ALTERNATE-KEY "Tools Menu".
END.

/* The POSITION X x Y y WIDTH dx HEIGHT dy call positions the open help window
   as specified */
ON CHOOSE OF b_pos IN FRAME x DO:
  SYSTEM-HELP helpfile POSITION X 2 Y 2 WIDTH 450 HEIGHT 450.
END.

/* The POSITION MAXIMIZE call maximizes the open help window as specified*/
ON CHOOSE OF b_max IN FRAME x DO:
  SYSTEM-HELP helpfile POSITION MAXIMIZE.
END.

/* The QUIT call causes the help engine to terminate, unless another
   application is using help. */
ON CHOOSE OF b_quit IN FRAME x DO:
  SYSTEM-HELP helpfile QUIT.
  RETURN.
END.

WAIT-FOR GO OF FRAME x.
```

See also

FILE-INFO system handle, SEARCH function

TENANT-ID function

Returns the tenant ID (as an integer) of the real tenant associated with a database connection.

Syntax

```
TENANT-ID ( [ database-name ] )
```

database-name

A character expression that evaluates to a logical database name or database alias. If the database is not enabled for multi-tenancy, the function returns zero (0).

If no database is specified and more than one database is connected, the AVM raises an error. If the database name is not a valid name for a connected database, the AVM raises an error.

Notes

- The tenant ID of the default tenant is `0`. The tenant ID of a non-default regular tenant is greater than `0` and that of a super tenant is less than `0`.
- For a super-tenant connection identity, this function returns the tenant ID of the super tenant, **not** the effective tenant.

See also

SET-EFFECTIVE-TENANT function, TENANT-NAME function, TENANT-NAME-TO-ID function

TENANT-NAME function

Returns the name (as a character string) of the real tenant associated with a database connection.

Syntax

```
TENANT-NAME ( [ database-name ] )
```

database-name

A character expression that evaluates to a logical database name or database alias. If the database is not enabled for multi-tenancy, the function returns the empty string ("").

If no database is specified and more than one database is connected, the AVM raises an error. If the database name is not a valid name for a connected database, the AVM raises an error.

Notes

For a super-tenant connection identity, this function returns the tenant name of the super tenant, **not** the effective tenant.

See also

SET-EFFECTIVE-TENANT function, TENANT-ID function, TENANT-NAME-TO-ID function

TENANT-NAME-TO-ID function

Returns the tenant ID associated with an input tenant name.

Syntax

```
TENANT-NAME-TO-ID ( tenant-name [ , database-name ] )
```

tenant-name

A character expression that evaluates to the name of a tenant.

database-name

A character expression that evaluates to a logical database name or database alias. If no database is specified and more than one database is connected, the AVM raises an error. If the database name is not a valid name for a connected database, the AVM raises an error.

Notes

- If a super tenant executes this function and *tenant-name* is not valid, the AVM raises an error.
- If a regular tenant executes this function, *tenant-name* must be the user's own tenant name or the AVM raises an error.

See also

TENANT-ID function, TENANT-NAME function

TERMINAL function

In Windows, in graphical interfaces, TERMINAL returns WIN3. In Windows, in character interfaces, TERMINAL returns CO80, BW80, or MONO, depending on the monitor type. On UNIX, TERMINAL returns the value of the $TERM environment variable. In batch mode, TERMINAL returns a null string.

Note: Does not apply to SpeedScript programming.

Syntax

```
TERMINAL
```

Example

This one-line procedure displays the type of terminal you are using:

r-term.p

```
MESSAGE "You are currently using a" TERMINAL "terminal.".
```

See also

TERMINAL statement

TERMINAL statement

Changes terminal type during program execution. On UNIX, changes the value of the TERM environment variable.

Note: Does not apply to SpeedScript programming.

Syntax

```
TERMINAL = termid
```

termid

A terminal type string. The *termid* can also be an expression. The AVM returns an error message if *termid* is not defined in the PROTERMCAP file. However, *termid* can be the word TERMINAL. The line TERMINAL=TERMINAL reinitializes the terminal.

Example

This procedure changes the terminal screen width from 80 columns to 132 columns, then back again:

r-seterm.p

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.
END.

TERMINAL = "wy60w".
OUTPUT TO TERMINAL PAGED.
FOR EACH Customer NO-LOCK:
  DISPLAY Customer WITH WIDTH 132.
END.

OUTPUT CLOSE.
TERMINAL = "wy60".
DISPLAY "Back to 80 columns." WITH CENTERED.
```

Notes

- TERMINAL does not change the physical characteristics of a terminal. You must supply a valid terminal type for the existing terminal state.
- The TERMINAL statement reinitializes the function key definitions based on the specified PROTERMCAP entry. If you have used ON statements to change function key definitions, the TERMINAL statement overrides those changes.
- If a subprocedure uses a frame, the frame is composed with the width that was in effect when the subprocedure was compiled. Changing the width (terminal type) outside the scope of that procedure will not change the frame width inside the procedure unless it is recompiled.

  The following sequence of statements does not work as intended, because `subp.p` is not recompiled before its second execution:

r-setrm1.p

```
TERMINAL = "wy60w".
RUN subp.p.

TERMINAL = "wy60".
RUN subp.p.

DISPLAY "Frame (132) too big for screen (80)" WITH CENTERED.
```

See also

TERMINAL function

THIS-OBJECT statement

A statement that invokes an overloaded constructor as the first statement in another overloaded constructor within the same defining class.

Syntax

```
THIS-OBJECT ( [ parameter [ , parameter ] ... ] )
```

([*parameter* [, *parameter*] ...])

Specifies zero or more parameters passed to the specified overloaded constructor. You must match the parameters identified by the called constructor with respect to number, data type, and mode, and you must specify sufficient information for each *parameter* to disambiguate the specified constructor from all other constructors in the class. Otherwise, ABL raises an error identifying the ambiguity.

For information on the parameter passing syntax and disambiguating overloaded constructors, see the Parameter passing syntax reference entry.

Note

If you invoke an overloaded constructor in the defining class using THIS-OBJECT, if the constructor that you invoke does not, itself, invoke another overloaded constructor in the defining class, it must invoke a constructor in the immediate super class, either implicitly or explicitly using the SUPER statement.

See also

CONSTRUCTOR statement, Parameter passing syntax, SUPER statement, THIS-OBJECT system reference

---

THIS-OBJECT system reference

An object reference to the currently running class instance that you can optionally use to access class members defined within the current class hierarchy.

Syntax

```
THIS-OBJECT [ : class-member-reference ]
```

class-member-reference

A reference to an instance variable data member, instance property, or instance method defined within the current class hierarchy. Any variable data member cannot be defined as an array (with an EXTENT).

Example

The following code fragment shows two instance methods defined within a class, where you must qualify a call to one of them with THIS-OBJECT because its name, `Display`, is an ABL reserved keyword:

```
CLASS TestClass:

  METHOD VOID Display( ):

  END METHOD.

  METHOD VOID Foo( ):

    THIS-OBJECT:Display( ).

  END METHOD.

END CLASS.
```

Notes

- You can use THIS-OBJECT without a method call to pass an object reference to the currently running object instance as a parameter, or to return an object reference to itself as a method return value.
- Use of the THIS-OBJECT system reference to call instance methods that are available within the class hierarchy where they are defined is generally optional. However, you **must** use the THIS-OBJECT system reference to call any instance method whose name is identical to an ABL reserved keyword. The same is true if you are calling an event method on a class-defined or inherited instance event of a class whose name is identical to an ABL reserved keyword. For more information, see the Class-based method call reference entry and the Class Events Reference.
- Use of the THIS-OBJECT system reference to access instance variable data members or properties that are available within the class hierarchy where they are defined is optional. However, you **must** use the THIS-OBJECT system reference to access any such data member or property whose name is identical to an ABL reserved keyword. For more information, see the Class-based data member access and Class-based property access reference entry.
- You **cannot** use the THIS-OBJECT system reference to access a static class member.
- To invoke other constructors defined within the current class, use the THIS-OBJECT statement.

See also

Class-based method call, THIS-OBJECT statement

TIME function

Returns an INTEGER value representing the time as the number of seconds since midnight. Use this function together with the STRING function to produce the time in hours, minutes, and seconds.

Syntax

```
TIME
```

Example

In `r-time.p`, the timeleft variable is set to the result of the TIME function subtracted from the number of seconds in a day. The procedure translates this value into seconds, minutes, and hours.

r-time.p

```
DEFINE VARIABLE hour     AS INTEGER NO-UNDO.
DEFINE VARIABLE minute   AS INTEGER NO-UNDO.
DEFINE VARIABLE sec      AS INTEGER NO-UNDO.
DEFINE VARIABLE timeleft AS INTEGER NO-UNDO.

timeleft = (24 * 60 * 60) - TIME.

/* seconds till next midnight */
sec = timeleft MOD 60.
timeleft = (timeleft - sec) / 60.

/* minutes till next midnight */
minute = timeleft MOD 60.

/* hours till next midnight */
hour = (timeleft - minute) / 60.

DISPLAY "Time to midnight:" hour  minute  sec .
```

This DISPLAY statement displays the current time.

r-time2.p

```
DISPLAY STRING(TIME,"HH:MM:SS").
```

See also

ADD-INTERVAL function, DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, ETIME function, INTERVAL function, ISO-DATE function, MONTH function, MTIME function, NOW function, TIME-SOURCE attribute, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

TIMEZONE function

Returns an INTEGER value representing the time zone offset from Coordinated Universal Time (UTC), in minutes. Use this function together with the STRING function to produce the time in hours, minutes, and seconds.

Note: Coordinated Universal Time (UTC) is the current universal standard for time. Local time zone values are relative to UTC. For example, Eastern Standard Time is UTC-05:00.

Syntax

```
TIMEZONE ( [ datetime-tz-expression | char-expression ] )
```

datetime-tz-expression

An expression whose value is a DATETIME-TZ.

char-expression

A character expression representing the time zone offset. The format of the expression must be +HH:MM.

If the TIMEZONE function has no arguments, it returns the client or server machine that serves as the time source for applications running during the ABL session (specified by the TIME-SOURCE attribute).

Example

Following is an example of using the TIMEZONE function:

```
DEFINE VARIABLE v-dt-tz AS DATETIME-TZ NO-UNDO
  INITIAL 2002-05-05T07:15:03.002-05:00.
DEFINE VARIABLE v-tz    AS INTEGER     NO-UNDO.

ASSIGN
  v-tz = TIMEZONE("+08:00")  /* v-tz = 480 */
  v-tz = TIMEZONE (v-dt-tz). /* v-tz = -300 */
```

See also

ADD-INTERVAL function, DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, ETIME function, INTERVAL function, ISO-DATE function, MONTH function, MTIME function, NOW function, TIME function, TIME-SOURCE attribute, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

TODAY function

Returns the current system date.

Syntax

```
TODAY
```

Example

This procedure prints the date in the first line at the top of each page of a report. Instead of using TODAY in the FORM statement, the procedure uses a variable to hold the date. This ensures that the same date appears on all pages of the report, even if this procedure runs through midnight.

r-today.p

```
DEFINE VARIABLE rptdate AS DATE NO-UNDO.

OUTPUT TO PRINTER.

rptdate = TODAY.
FORM HEADER rptdate "Customer List" AT 34
  "Page" AT 66 PAGE-NUMBER FORMAT ">>>9" SKIP(2)
  WITH NO-BOX PAGE-TOP.
VIEW.

FOR EACH Customer NO-LOCK:
  DISPLAY Customer.Name AT 1 Customer.Address AT 31
    Customer.City + ", " + " " + Customer.State FORMAT "x(35)" AT 31
    WITH NO-BOX NO-LABELS CENTERED.
END.
```

PAGE-TOP frames are re-evaluated on every new page. Therefore, if you do not use a variable for the date, a different date is displayed on the following page(s) if the report starts before midnight and ends after midnight.

See also

ADD-INTERVAL function, DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, ETIME function, INTERVAL function, ISO-DATE function, MONTH function, MTIME function, NOW function, TIME function, TIME-SOURCE attribute, TIMEZONE function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

TO-ROWID function

Converts a string representation of a ROWID to a valid ROWID value.

Syntax

```
TO-ROWID ( rowid-string )
```

rowid-string

A string representation of a ROWID. Since ROWID values are a variable sequence of hexadecimal digits, *rowid-string* must be in the form "0x*hex-digits*", where *hex-digits* is any string of characters from 0 through 9 and A through F.

Example

The following procedure (`r-torwid.p`) selects Customer `Balance` and credit information and displays it in a browse. You can select any number of rows to store and display more information on the selected Customers.

r-torwid.p

```
DEFINE VARIABLE hcustb AS HANDLE  NO-UNDO.
DEFINE VARIABLE irow   AS INTEGER NO-UNDO.

DEFINE QUERY custq FOR Customer.
DEFINE BUFFER cust2 FOR Customer.

DEFINE TEMP-TABLE rowtab FIELD rowchar AS CHARACTER
  INDEX rowi IS UNIQUE PRIMARY rowchar ASCENDING.

DEFINE BROWSE custb QUERY custq
  DISPLAY Customer.CustNum Customer.Name Customer.Balance
    Customer.CreditLimit WITH 10 DOWN MULTIPLE.

DEFINE BUTTON bstore   LABEL "Store Selections".
DEFINE BUTTON bdisplay LABEL "Display Call Selections".
DEFINE BUTTON bclear   LABEL "Clear Storage".

DEFINE FRAME brs-frame custb SKIP bstore bdisplay bclear.
DEFINE FRAME dsp-frame cust2.CustNum cust2.Name cust2.Phone
  WITH 5 DOWN SCROLL 1.

ON CHOOSE OF bstore DO:
  DO irow = 1 TO custb:NUM-SELECTED-ROWS:
    IF custb:FETCH-SELECTED-ROW(irow) AND
      NOT CAN-FIND(rowtab WHERE STRING(ROWID(Customer)) = rowchar) THEN DO:
      CREATE rowtab NO-ERROR.
      ASSIGN rowchar = STRING(ROWID(Customer)) NO-ERROR.
    END.
  END.
END.

ON CHOOSE OF bdisplay DO:
  CLEAR FRAME dsp-frame ALL.
  FOR EACH rowtab WITH FRAME dsp-frame:
    FIND cust2 WHERE ROWID(cust2) = TO-ROWID(rowchar).
    DISPLAY cust2.CustNum cust2.Name cust2.Phone.
    DOWN WITH FRAME dsp-frame.
  END.
END.

ON CHOOSE OF bclear DO:
  IF custb:DESELECT-ROWS() THEN
  FOR EACH rowtab:
    DELETE rowtab.
  END.
  FRAME dsp-frame:VISIBLE = FALSE.
END.

OPEN QUERY custq PRESELECT EACH Customer.
ENABLE ALL WITH FRAME brs-frame.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Thus, when you choose the bstore button, `r-torwid.p` stores the ROWID string values of all selected Customer records in a temp-table. When you choose the bdisplay button, it displays the selected Customer Phone information in a DOWN frame by converting each stored ROWID string to a ROWID value and finding the corresponding Customer record. (The example also allows you to add selections and restart by deleting the existing selections.)

Note

Although TO-ROWID converts a properly formatted string to a ROWID value, there is no guarantee that this value corresponds to an existing record in your database.

See also

DATE function, DECIMAL function, INTEGER function, ROWID function, STRING function

---

TRANSACTION function

Returns a LOGICAL value that indicates whether a transaction is currently active.

Syntax

```
TRANSACTION
```

See also

Transaction object handle

TRANSACTION-MODE AUTOMATIC statement
(PAS for OE only)

Causes the procedure file that executes this statement to become an automatic transaction initiating procedure. This transaction initiating procedure allows you to control an automatic transaction in the context of an application server session.

Note: Does not apply to SpeedScript programming.

Syntax

```
TRANSACTION-MODE AUTOMATIC [ CHAINED ]
```

`CHAINED`

Tells the application server session to automatically create a new transaction every time the current transaction is either committed or rolled back.

Notes

- This statement must appear before any other executable statement in a top-level persistent procedure (transaction initiating procedure) running on the application server.
- You can control an automatic transaction by accessing the attributes and methods of the transaction object. You can access these attributes and methods on the transaction handle returned by the TRANSACTION attribute of any application server procedure handle.

- An automatic transaction remains open in an application server session until:
  - The current request service returns control to the client after an application server procedure invokes the transaction handle SET-COMMIT( ) method or SET-ROLLBACK( ) method.
  - The transaction initiating procedure is deleted from the session.
- If you specify the CHAINED option, a transaction is always active in the application server session until either the transaction initiating procedure is deleted or the application server session terminates.
- If you do not specify the CHAINED option and the transaction initiating procedure is still active, after the current transaction terminates, a client application can start a new transaction by directly calling any remote internal procedure of the transaction initiating procedure. When so executed, this remote internal procedure (which can otherwise be empty) creates a new transaction that you can control using the transaction handle.
- As long as an automatic transaction is open, you can execute any internal procedure of the current transaction initiating procedure from any other procedure running on the application server. However, if no automatic transaction is open, only a client application can execute such an internal procedure as a remote procedure call, which then opens an automatic transaction. If an application server procedure tries to execute such an internal procedure with no automatic transaction open, the procedure call returns an error.
- If a transaction is open when you delete the transaction initiating procedure, the transaction is committed or rolled back according to the value of the transaction handle DEFAULT-COMMIT attribute.

See also

Transaction object handle

Trigger phrase

Defines triggers on one or more user-interface events for a single user-interface component. Use the Trigger phrase within the statement that defines or creates the associated user-interface component.

Note: Does not apply to SpeedScript programming.

Syntax

```
TRIGGERS:
  { ON event-list [ ANYWHERE ]
      {     trigger-block
        |  PERSISTENT RUN procedure
             [ IN handle ]
             [ ( input-parameters ) ]
      }
  } ...
  END [ TRIGGERS ]
```

event-list

The event or events with which the trigger block is associated. To specify more than one event, separate them with commas as follows:

```
event1 , event2 ...
```

The events you can specify depend on the type of the associated widget. See the reference entry for the appropriate widget. For more information on each user interface event that ABL supports, see the Handle-based Object Events Reference.

ANYWHERE

Specifies that the trigger is a group trigger. This means that it applies not only to the widget being defined or created, but also is a default to any widget contained within that widget. This allows you to create a default trigger for all widgets in a frame or window. You can override the group trigger by defining a trigger on the same event specifically for the widget (or by defining a group trigger on an intervening widget).

trigger-block

A sequence of ABL statements to be executed when any of the specified events occur. The trigger block must be a single ABL statement or a DO block.

PERSISTENT RUN *procedure* [IN *handle*] [(*input-parameters*)]

Specifies a persistent trigger; that is, a trigger that remains in effect after the current procedure terminates. A persistent trigger must be a procedure specified by *procedure*. The trigger procedure can take one or more input parameters; it cannot have any output parameters. The parameters are evaluated **when the trigger is defined**. They are **not** re-evaluated each time the trigger executes.

If you specify the IN *handle* option, *procedure* must be the name of an internal procedure defined in the external procedure specified by *handle*, where *handle* is an expression that evaluates to a valid procedure handle. The external procedure must be in scope when you run *procedure*.

Example

This procedure defines triggers for two buttons:

r-trigp.p

```
DEFINE FRAME cust-frame.
DEFINE QUERY custq FOR Customer.

DEFINE BUTTON nextcust LABEL "Next"
  TRIGGERS:
    ON CHOOSE DO:
      GET NEXT custq.
      DISPLAY Customer WITH FRAME cust-frame.
    END.
  END TRIGGERS.

DEFINE BUTTON prevcust LABEL "Previous"
  TRIGGERS:
    ON CHOOSE DO:
      GET PREV custq.
      DISPLAY Customer WITH FRAME cust-frame.
    END.
  END TRIGGERS.

OPEN QUERY custq FOR EACH Customer.
GET FIRST custq.
DISPLAY Customer WITH FRAME cust-frame.
ENABLE nextcust AT COLUMN 1 ROW 7 prevcust WITH FRAME cust-frame.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- If you specify the Trigger phrase in the definition of a user-interface component, the Trigger phrase must be the last option in the component definition.
- If you specify a trigger when you define a widget then that trigger applies to every instance of that widget. For example, in `r-trigp.p`, if you enable the nextcust button in more than one frame, each of those buttons inherits the nextcust trigger.
- The input parameters for a persistent trigger are evaluated when the trigger is attached. (For the Trigger phrase, the trigger is attached when the widget is realized.) This means, for example, that you cannot pass the SELF handle as an input parameter.
- The external procedure specified by *handle* is in scope if it is the current procedure, a procedure on the call stack, or a persistent procedure.

See also

CREATE widget statement, DEFINE MENU statement, ON statement

TRIGGER PROCEDURE statement

Defines a schema trigger.

Syntax

```
TRIGGER PROCEDURE FOR event OF object [ options ]
```

event

The event for which the schema trigger is being defined. The valid events are CREATE, DELETE, FIND, WRITE, and ASSIGN.

object

The object on which the event is defined. If the event is CREATE, DELETE, FIND, or WRITE, the object must be a reference to a database table. If the event is ASSIGN, the object must be a reference to a database field qualified by a table name.

options

Optional parts of the trigger header.

Headers for CREATE, DELETE, and FIND triggers take no options. Their syntaxes are as follows:

```
TRIGGER PROCEDURE FOR CREATE OF table
```

```
TRIGGER PROCEDURE FOR DELETE OF table
```

```
TRIGGER PROCEDURE FOR FIND OF table
```

In the header for a WRITE trigger you can optionally include one or two buffer names.

```
TRIGGER PROCEDURE FOR WRITE OF table
  [ NEW [ BUFFER ] buffer-name1 ]
  [ OLD [ BUFFER ] buffer-name2 ]
```

For an ASSIGN trigger procedure, there are two ways you can specify the field/value for validation. The first way is to use the OF phrase and the second way is to use the NEW [VALUE] phrase. If you use the OF phrase, the value from the ASSIGN has been put into the buffer, so you can check it from there. However, the change has not been committed yet, so if this trigger returns error, the new value will be backed out. If you use the NEW VALUE phrase, then the new value is available for validation via the *value1* variable. Each way accomplishes the same task; it's a matter of coding preference.

```
TRIGGER PROCEDURE FOR ASSIGN
  {     { OF table .field }
    |   { NEW [ VALUE ] value1
            { AS data-type | LIKE db-field }
        }
  }
  [ COLUMN-LABEL label ]
  [ FORMAT format-string ]
  [ INITIAL constant ]
  [ LABEL label-string ]
  [ NO-UNDO ]
  [ OLD [ VALUE ] value2
     { AS data-type | LIKE db-field  }
     [ COLUMN-LABEL label ]
     [ FORMAT format-string ]
     [ INITIAL constant ]
     [ LABEL label-string ]
     [ NO-UNDO ]
  ]
```

Examples

The following is a WRITE trigger for the Customer table. It uses the OLD BUFFER option so that it can determine whether the CustNum value has changed. If the Customer's outstanding balance exceeds its Credit Limit, the trigger returns the error condition (in which case the record is not updated).

r-wrcust.p

```
TRIGGER PROCEDURE FOR WRITE OF Customer OLD BUFFER oldCustomer.

/* Variable Definitions */
DEFINE VARIABLE ix          AS INTEGER NO-UNDO.
DEFINE VARIABLE Outstanding AS INTEGER NO-UNDO.

/* Check to see if the user changed the Customer Number */
IF Customer.CustNum <> oldCustomer.CustNum
  AND oldCustomer.CustNum <> 0 THEN DO:
  /* If the user changed the Customer Number, find all related Orders and
     change their Customer numbers. */
  FOR EACH order OF oldCustomer:
    Order.CustNum = Customer.CustNum.
    ix = ix + 1.
  END.
  IF ix > 0 THEN
    MESSAGE ix "Orders changed to reflect the new Customer number!".
END.

/* Ensure that the Credit Limit value is always greater than the sum of
   this Customer's outstanding balance */
FOR EACH Order OF Customer NO-LOCK:
  FOR EACH OrderLine OF Order NO-LOCK:
    Outstanding = Outstanding + (OrderLine.Qty * OrderLine.Price).
  END.
END.
FOR EACH Invoice OF Customer NO-LOCK:
  Outstanding = Outstanding +
    (Invoice.Amount - (Invoice.TotalPaid + Invoice.Adjustment)).
END.

IF Customer.CreditLimit < Outstanding THEN DO:
  MESSAGE "This Customer has an outstanding balance of: " Outstanding
    ".  The Credit Limit MUST exceed this amount!".
  RETURN ERROR.
END.
```

The following is an example of an ASSIGN trigger using the OF phrase.

```
TRIGGER PROCEDURE FOR ASSIGN OF Customer.NAME.

IF Customer.NAME = "Customer X" THEN

DO:

   MESSAGE "Customer X not allowed."

       VIEW-AS ALERT-BOX INFORMATION BUTTONS OK.

   RETURN ERROR.

END.
```

The following is an example of an ASSIGN trigger using the NEW [VALUE] phrase.

```
TRIGGER PROCEDURE FOR ASSIGN NEW VALUE value1 AS CHARACTER.

IF value1 = "Customer X" THEN

DO:
```

```
   MESSAGE "Customer X not allowed."
       VIEW-AS ALERT-BOX INFORMATION BUTTONS OK.
   RETURN ERROR.
END.
```

Notes

- Use the Data Dictionary to associate a trigger procedure with a table or field in the database.
- Some 3GL applications execute schema triggers. Triggers might also be executed in batch mode. Therefore, you should avoid any user-interface interactions within schema trigger procedures.
- For a `TRIGGER PROCEDURE FOR WRITE` the `OLD BUFFER` does not contain a before image for CLOB or BLOB columns.

See also

PROCEDURE statement

TRIM function

Removes leading and trailing white space, or other specified characters, from a CHARACTER or LONGCHAR expression. The data type of the returned value matches the data type of the expression passed to the function.

Syntax

```
TRIM ( expression [ , trim-chars ] )
```

expression

An expression (a constant, field name, variable name, or expression) whose value is a CHARACTER or LONGCHAR. If *expression* is a case-sensitive variable, the AVM performs a case-sensitive trim. If *expression* is a LONGCHAR, the result is in the same code page.

trim-chars

A character expression that specifies the characters to trim from *expression*. If you do not specify *trim-chars*, the TRIM function removes spaces, tabs, line feeds, and carriage returns.

Examples

The following procedure displays a menu that you can use to display Customer and Order information. The option numbers are displayed with leading spaces. The TRIM function removes the leading white space so the menu selection can be easily evaluated.

r-trim.p

```
DEFINE VARIABLE menu AS CHARACTER NO-UNDO EXTENT 3.

DO WHILE TRUE:
  DISPLAY
    "    1.    Display Customer Data" @ menu[1] SKIP
    "    2.    Order Data" @ menu[2] SKIP
    "    3.    Exit" @ menu[3] SkIP
    WITH FRAME choices NO-LABELS.
  CHOOSE FIELD menu AUTO-RETURN WITH FRAME choices
    TITLE "Demonstration Menu" CENTERED ROW 10.
  HIDE FRAME choices.
  IF TRIM(FRAME-VALUE) BEGINS "1" THEN RUN r-dblnkc.p.
  IF TRIM(FRAME-VALUE) BEGINS "2" THEN RUN r-dblnko.p
  IF TRIM(FRAME-VALUE) BEGINS "3" THEN LEAVE.
END.
```

The following example reads a text file and breaks it into words. It assumes that all words are separated by at least one space character. It uses the TRIM function with one parameter to remove white space from the ends of each input line. It then uses the TRIM function with two parameters to remove any punctuation characters from each word.

r-trim2.p

```
DEFINE VARIABLE infile     AS CHARACTER NO-UNDO FORMAT "x(60)"
  LABEL "Input File".
DEFINE VARIABLE intext     AS CHARACTER NO-UNDO.
DEFINE VARIABLE next-space AS INTEGER   NO-UNDO.
DEFINE VARIABLE word       AS CHARACTER NO-UNDO FORMAT "x(32)"
  LABEL "Words".

/* Get the name of a text file and set input to that file. */.
SET infile.
INPUT FROM VALUE(infile).

DO WHILE TRUE:
  /* Read the next line from the file. */
  IMPORT UNFORMATTED intext.
  intext = TRIM(intext).

  DO WHILE TRUE:
    /* Find the next space character. If none found, find the end of
       string. */
    next-space = INDEX(intext, " ").
    IF next-space = 0 THEN
      next-space = LENGTH(intext) + 1.

    /* If the string contains no (more) words, then read the next line. */
    IF next-space = 1 THEN LEAVE.

    /* Pull the first word off the string. Remove any punctuation characters
       around it. */
    word   = SUBSTRING(intext, 1, next-space - 1).
    word   = TRIM(word, ",.;:!? ~"~ '[]()").
    intext = TRIM(SUBSTRING(intext, next-space + 1)).

    /* Display the word. */
    DISPLAY word WITH DOWN FRAME x.
    DOWN WITH FRAME x.
  END.
END.
```

Notes

- The TRIM function is double-byte enabled. The specified *expression* and *trim-chars* arguments can contain double-byte characters. TRIM does not remove double-byte space characters by default.
- A character string displays with the default format of x(8), unless you specify a format or use a statement such as DISPLAY @ *literal*.
- You can use the DEBLANK option of the Format phrase to remove leading spaces for fields in the input buffer.
- If *expression* is a case-sensitive field or variable, then *trim-chars* is also case sensitive. Otherwise, *trim-chars* is not case sensitive.

See also

LEFT-TRIM function, RIGHT-TRIM function

TRUNCATE function

Truncates a decimal expression to a specified number of decimal places, returning a decimal value.

Syntax

```
TRUNCATE ( expression , decimal-places )
```

expression

A decimal expression that you want to truncate.

decimal-places

A non-negative integer expression that indicates the number of decimal places for a truncated *expression*.

Example

This procedure doubles each Customer's `CreditLimit` and then truncates that value before rounding it to the nearest $1000:

r-trunc.p

```
FOR EACH Customer:
  FORM Customer.CustNum Customer.Name Customer.CreditLimit
    new-max LIKE Customer.CreditLimit LABEL "New Credit limit".
  DISPLAY Customer.CustNum Customer.Name Customer.CreditLimit.
  Customer.CreditLimit =
    TRUNCATE((Customer.CreditLimit * 2) / 1000 ,0) * 1000.
  IF Customer.CreditLimit < 15000 THEN
    Customer.CreditLimit = 15000.
  DISPLAY Customer.CreditLimit @ new-max.
END.
```

Notes

You can use the TRUNCATE function to treat division as integer division. For example, i = TRUNCATE (x / y, 0).

See also

ROUND function

Type-name syntax

Specifies the name of an ABL or .NET data type that you can specify as a single instance or as an array of such instances. Thus, an ABL data type can be a built-in primitive type or an array of such primitive types, or it can be a class-based built-in or user-defined object type (such as a class, enumeration, or interface type) or an ABL array of such object types. A .NET data type can only be a class-based object type (such as a class, structure, enumeration, or interface) or an ABL array of such object types. The EXTENT option, used to define an array of types, is thus counted as part of the array type name.

Syntax

```
{ primitive-type-name | object-type-name } [ EXTENT [ constant ] ]
```

You use a different syntax when defining variables using the VAR statement:

```
primitive-type-name [ [ [ constant ] ] ] | [ CLASS ] object-type-name [ [
[ constant ] ] ]
```

primitive-type-name

An unquoted string that specifies the name of a built-in ABL primitive type. For information on the supported primitive types and their names, see the Data types reference entry.

object-type-name

Specifies an ABL or .NET object type using the following syntax:

```
[ " ] ABL-object-type [ " ] |
[ " ] dotNET-object-type [ " ]
```

```
[ " ] ABL-object-type [ " ]
```

An ABL object type, where *ABL-object-type* consists of text elements with the following syntax:

```
[ package-name . ] class-interface-or-enum-name
```

package-name

A period-separated list of text components that, along with *class-interface-or-enum-name*, uniquely identify an ABL class, interface, or enum. These text components specify a package that is based on a valid directory pathname, relative to PROPATH, which identifies the location of the file that defines the class, interface, or enum. Thus, each text component of *package-name* maps to a directory level in the path, and each slash separator in the path corresponds to a period separating two components.

If specified, the relative path of the class definition file represented by *package-name* must remain constant between compile time and run time. If the class definition file resides directly on PROPATH, the class or interface is not defined in a package and therefore has no *package-name* in its type name.

With the presence of an appropriate USING statement you can also specify an ABL object type that is defined in a package using the *class-interface-or-enum-name* without its qualifying *package-name*. For more information, see the notes for this reference entry.

class-interface-or-enum-name

The name of an ABL class, interface, or enumeration. This name must match the name of a class definition file (excluding the `.cls` or `.r` extension) located in the relative path represented by *package-name*, if specified. The name must begin with an alphabetic character and it cannot contain a period or a space.

If the *ABL-object-type* has a *package-name* that contains embedded spaces, you must enclose the entire *ABL-object-type* in quotes (`"`). Otherwise, quotes are optional.

Note: Do not place a class definition file in a directory whose name contains a period (`.`) character; ABL interprets the component after the period as another directory level and will therefore not find the referenced class definition file.

["] *dotNET-object-type* **["]**

A .NET object type, where *dotNET-object-type* consists of text elements with the following syntax:

```
[ namespace . ] dotNET-object-name  [ + inner-name ]
```

namespace

A period-separated list of text components that, along with *dotNET-object-name*, uniquely identify a .NET type. The components of *namespace* are defined according to .NET requirements. ABL does not support access to .NET types defined in the default namespace. In other words, you cannot access a .NET type that does not have a *namespace* defined for it.

However, with the presence of an appropriate USING statement, you can also specify a .NET object type using the *dotNET-object-name* without its qualifying *namespace*. For more information, see the notes for this reference entry.

dotNET-object-name

The name of a .NET class (including a structure or enumeration), delegate, interface, or other object type referenced within the .NET namespace specified by *namespace*.

inner-name

The name of a .NET nested (inner) type defined within the .NET type definition specified by *namespace.dotNET-object-name*. Thus, *inner-name* can represent the name of an:

- Inner class defined by the specified .NET class
- Inner enumeration defined by the specified .NET class
- Inner interface defined by the specified .NET interface

Caution: .NET languages normally separate the name of an inner type from the defining type name using a period (.). You must replace this period with a plus sign (+) to reference the inner type name in ABL.

If the name part of the *dotNET-object-type* contains any embedded spaces, square brackets ([]), or angle brackets (<>) you must enclose the entire *dotNET-object-type* in quotes ("). Otherwise, quotes are optional.

For more information on defining object types in ABL, see the CLASS statement or the INTERFACE statement reference entry, depending on the type of class-based object.

EXTENT **[** *constant* **]**

EXTENT indicates that it is an array of the specified type and *constant* is an integer value that specifies the number of elements in the array. Without *constant*, the array is an indeterminate array type.

For more information on defining ABL data elements with primitive or object types, see the DEFINE PARAMETER statement, the DEFINE PROPERTY statement, the DEFINE TEMP-TABLE statement, the DEFINE VARIABLE statement, the DEFINE WORK-TABLE statement, or the Parameter definition syntax reference entry.

Examples

If your PROPATH is "C:/myfiles", and your class definition file name is "C:/myfiles/acme/myObjs/CustObjs.cls", then ABL requires *package-name* to be "acme.myObjs." and *class-or-interface-name* to be "CustObjs".

The .NET Button class is in the System.Windows.Forms namespace. Therefore, you reference its qualified (complete) type name like this:

```
System.Windows.Forms.Button
```

The .NET ControlCollection class is an inner class of System.Windows.Forms.Control. Therefore, you reference its qualified type name like this:

```
System.Windows.Forms.Control+ControlCollection
```

The following code fragment defines object references to the .NET type, `System.Drawing.Point`, and to a one-dimensional .NET array of `System.Drawing.Point` elements using `DEFINE VARIABLE` statements:

```
DEFINE VARIABLE rPoint      AS CLASS System.Drawing.Point NO-UNDO.
DEFINE VARIABLE rPointArray AS CLASS "System.Drawing.Point[]" NO-UNDO.
```

The following code fragment defines the same variables using VAR statements:

```
VAR CLASS System.Drawing.Point rPoint.
VAR CLASS "System.Drawing.Point[]" rPointArray.
```

Notes

- Both elements of an ABL object type name (*package-name* and *class-interface-or-enum-name*) must conform to the case sensitivity requirements of the operating system (e.g., UNIX or Windows). On a case-sensitive OS, only the first reference to the object type name must be case correct. ABL follows this initial letter case for all subsequent references to the type.
- You cannot specify `Progress` as the first component of *package-name* for any ABL user-defined class. For example, `Progress.Inventory.UpdateInv` is an invalid type name for a user-defined class and results in a compiler error.
- You must use equivalent static type-name syntax to reference static members of a class whose names are identical to reserved keywords, as well as from **inside** the ABL class hierarchy, including when you call event methods on inherited static class events.

  For more information on using static type-name syntax, see the appropriate reference entry for the static member: Class-based data member access, Class-based method call, or Class-based property access. Also, for information on calling event methods on a static event, see the Class Events Reference.
- ABL allows you to name an ABL class or interface using an ABL reserved keyword, such as Display or DISPLAY. For a list of ABL reserved keywords, see the Keyword Index. However, this is not a recommended coding practice, in part because ABL does not fully support the use of static type-name syntax for a class or interface name that is an ABL reserved keyword. This limitation is mitigated if the type name is a fully qualified type name that includes both the *package-name* and class name.
- Depending on the type definition, the context of the type reference, and the presence of an appropriate USING statement, you can use a qualified or an unqualified class, interface, or enum name to reference an ABL object type. A qualified type name is one that includes both a *package-name* and a *class-interface-or-enum-name*. An unqualified type name is one that includes *class-interface-or-enum-name* alone, without a *package-name*. All qualified type names must be fully qualified, using a complete *package-name*. ABL does not support partially qualified type names using a partial *package-name* specification. Without an appropriate USING statement, you can only specify an unqualified type name when the type is defined directly on PROPATH (not in a package). In this case, the unqualified type name is the complete type name for the object type.

  Note: For classes with static members, Progress Software Corporation recommends that you either define the class in a *package-name* and always reference its static members using the fully qualified class type name, or use a naming convention that defines static members with unique names.
- Depending on the presence of an appropriate USING statement, you can use a qualified or unqualified type name to reference a .NET type. A qualified .NET type name is one that includes both a *namespace* and a *dotNET-object-name*. An unqualified .NET type name is one that includes *dotNET-object-name* alone, without its defined *namespace*. All qualified .NET type names must be fully qualified, using a complete

namespace. ABL (unlike .NET languages) does not support partially qualified type names using a partial *namespace* specification. Without an appropriate USING statement, you cannot specify an unqualified .NET type name, because ABL does not support access to .NET types that are defined in the .NET default namespace.

- You can reference a .NET delegate type only in the context of a DEFINE EVENT statement.
- When you specify the type name of a class, interface, or enumeration in its ABL type definition statement (CLASS, INTERFACE, or ENUM statement), and the class file where the type is defined resides in a package directory, you must specify the qualified type name (*package-name.class-interface-or-enum-name*) for its definition, even with the presence of an appropriate USING statement. You can only specify an unqualified type name (*class-interface-or-enum-name* alone) in its type definition statement when the class file defining the type resides directly on PROPATH (is not in a package directory).
- ABL allows a locally scoped name (for example, a variable, temp-table, buffer name) to be identical to the name of an accessible class. If this is the case, the locally scoped name takes precedence over any static type-name reference to the unqualified class type name, causing a compiler error on any such reference to a static class member. To avoid this error, ensure that no locally scoped names have the same name as the class, and always either use fully qualified type-name syntax to reference the static class member or use naming conventions to guarantee uniqueness for static member names.
- Because of the similarity between the syntax of *object-type-name* references and table buffer and field references (including similarities between syntax type-name references and field attribute references), ABL can encounter ambiguous references among them, resulting in compilation errors. To ensure that ABL always recognizes *object-type-name* references unambiguously, either define all object type names with at least three period (`.`) separators (in other words, at least three components in any *package-name* specification) or use naming conventions to guarantee that *object-type-name* references and database element references are unique. Otherwise, ABL recognizes all such references as *object-type-name* references.

  For example, suppose that you have a user-defined class type in your PROPATH, `Sports2020.Customer.Name`, which defines a static property, `Label`. The reference in the following code fragment would then be ambiguous because ABL cannot distinguish between a reference to this static `Label` property and a reference to the LABEL attribute on the fill-in widget that is defined for the `Name` field in the `Customer` table of the `Sports2020` database:

```
DEFINE VARIABLE cLabel AS CHARACTER NO-UNDO.
DEFINE FRAME a Sports2020.Customer.Name.

FIND FIRST Sports2020.Customer.
cLabel = Sports2020.Customer.Name:Label. /* Ambiguous reference */
```

- For more information on the types provided by Microsoft .NET, see the .NET class library documentation available from the Microsoft® Development Network (MSDN) at http://msdn.microsoft.com/en-us/library/aa338209.aspx.

Note: The .NET class library does not list nested types as members (which they are) of the type that defines them. Instead, it lists each nested type definition immediately following the type that defines it. You can identify .NET nested types in this listing by the period (`.`) that separates the *inner-name* of the nested type name from the name of the .NET type that defines it. As shown in this syntax and examples, ABL uses a plus (+) instead of a period (`.`) to reference the *inner-name* of a nested type.

- To compile ABL references to a .NET type, the type must be defined in either an automatically-loaded .NET assembly or in a .NET assembly that is listed in an OpenEdge assembly references file named `assemblies.xml`. This file must be available either in the working directory or in the directory specified

by the Assemblies (`-assemblies`) startup parameter. You must also deploy this file at run time. For more information on working with assembly references files, see *Use GUI for .NET Objects in ABL Applications*. OpenEdge also loads the following assemblies automatically if they are not referenced in `assemblies.xml`:

- **`Progress.NetUI.dll`** — Assembly where all custom OpenEdge .NET classes and interfaces reside
- **`Mscorlib.dll`** — Assembly where all core Microsoft .NET classes and interfaces reside
- **`System.Windows.Forms.dll`** — Assembly where all Microsoft form and control classes and interfaces reside
- **`System.Drawing.dll`** — Assembly where basic Microsoft graphics classes and interfaces reside

The point at which OpenEdge loads any .NET assemblies depends on your application (see the following note on loading the CLR).

- OpenEdge loads the .NET Common Language Runtime (CLR) and all required assemblies for access by ABL if any of the following conditions exists:
  - You use the Preload CLR (`-preloadCLR`) startup parameter to start the current ABL session, which causes the .NET CLR to load at session startup.
  - The ABL compiler encounters a reference to a qualified object type name, and either that type name matches (or results from a match) to a USING statement containing the FROM ASSEMBLY option or the type cannot be found on PROPATH (meaning that ABL assumes it is a .NET type). This reference can occur on any ABL statement where a type name is used, such as a DEFINE VARIABLE, CAST, static method call, and so on.

    Note: An unqualified type name that does not resolve to a qualified type name from a USING statement match is assumed to be the name of an ABL user-defined type, which does not cause the CLR to load.

  - At run time, the ABL Virtual Machine (AVM) executes a statement using the NEW function (classes) for a .NET object.
  - At run time, the AVM accesses a static .NET method, property, or data member.
  - At run time, the AVM accesses the FIRST-FORM attribute or LAST-FORM attribute, or accesses the ACTIVE-FORM system reference.
- When you first reference a .NET type in a procedure or class file (compilable unit), ABL is always case-sensitive when using your reference to identify and search for the type definition. However, for names of .NET class members, including properties, data members, methods, and enumeration members, ABL is case insensitive.

  Note: Microsoft recommends that .NET names never be distinguished by letter case, alone. However, if a .NET class does have two class members with names distinguished only by letter case, ABL finds only the first one defined in the class.

- For information on how ABL maps .NET types to ABL primitive types, see the Data types reference entry.
- When creating and accessing .NET arrays in ABL, you must use methods of the .NET `System.Array` type. ABL also supports automatic mappings between ABL and one-dimensional .NET arrays, allowing supported .NET and ABL arrays to be assigned to each other or passed to each other as routine parameters. For more information, see the Data types reference entry.

- Note that you can specify an ABL array of .NET array objects, which is equivalent to a two-dimensional array. The following example defines an ABL array with 10 elements, each of which can reference a one-dimensional .NET array of `System.Drawing.Point` objects:

```
DEFINE VARIABLE rPointArray AS CLASS "System.Drawing.Point[]"
  EXTENT 10 NO-UNDO.
```

- You can also specify a constructed .NET generic type for *`dotNET-object-name`*. ABL supports references to a .NET generic type for all uses of .NET types except to specify a .NET class to inherit or a .NET interface to implement in an ABL class definition. For more information on constructing and using .NET generic types in ABL, see the Data types reference entry.

See also

Class-based data member access, Class-based method call, Class-based property access, CLASS statement, Data types, DEFINE VARIABLE statement, ENUM statement, INTERFACE statement, NEW function (classes), USING statement, VAR statement

---

TYPE-OF function

Verifies that the class instance to which the specified object reference points is an instance of the specified object type, inherits from the specified super class, or implements the specified interface. If the object reference points to an instance of the specified type, the function returns TRUE. Otherwise, it returns FALSE.

Syntax

```
TYPE-OF ( object-reference , object-type-name ).
```

`object-reference`

An object reference to a class instance.

`object-type-name`

Specifies the type name of a class, a super class, or an interface that might be defined, inherited from, or implemented (respectively) by the object referenced by *`object-reference`*. If *`object-type-name`* specifies an interface that is inherited by an interface that the class instance implements, then the inherited interface is also implemented by the class instance specified by the *`object-type-name`*. Specify the object type name using the syntax described in the Type-name syntax reference entry. With an appropriate USING statement, you can also specify an unqualified class or interface name alone.

See also

Type-name syntax, USING statement

UNBOX function
(Windows only; GUI for .NET only)

Unboxes a .NET `System.Object` or array object and returns a value of a corresponding ABL primitive or array type.

Syntax

```
UNBOX ( object-reference )
```

object-reference

Specifies an object reference to a boxed .NET primitive value (`System.Object`) or to a one-dimensional .NET array object. For a list of the .NET primitive types that a `System.Object` can box, see Table 24: Implicit mappings between .NET and ABL data types.

If *object-reference* points to a boxed .NET primitive value, the function returns a value of the ABL primitive type that implicitly maps to the boxed .NET type.

If *object-reference* points to a one-dimensional .NET array object, the function returns a copy of the elements contained by the .NET array as an ABL array. If these elements are .NET mapped object types, the returned ABL array contains elements of the corresponding ABL primitive type. Otherwise, the ABL array contains object reference elements of the actual object type contained by the input .NET array.

Example

The following code creates a .NET `DataTable` with a single `DataRow` containing two columns, an integer and a character string, and adds data to the two columns in that row. It then processes the data from the `"CustNum"` (`System.Int32`) column to get an indication if its value is even or odd and assigns the result to the ABL INTEGER variable `iVal`. The `Item` indexed property that is used to access the data has the data type `System.Object`. So, to use its value in an expression, you must use the UNBOX function to unbox the underlying .NET mapped data type of the `System.Object` value. In this case, the referenced `System.Object` represents a `System.Int32` value:

```
USING System.Data.* FROM ASSEMBLY.

DEFINE VARIABLE dataTable1 AS DataTable.
DEFINE VARIABLE dcCustNum  AS DataColumn.
DEFINE VARIABLE dcName     AS DataColumn.
DEFINE VARIABLE row1       AS DataRow.
DEFINE VARIABLE iVal       AS INTEGER.

dataTable1 = NEW DataTable( INPUT "Customer" ).

/* Create columns for a dataTable */
dcCustNum = NEW DataColumn( INPUT "CustNum" ).
dcName    = NEW DataColumn( INPUT "Name" ).
dataTable1:COLUMNS:ADD( INPUT dcCustNum ).
dataTable1:COLUMNS:ADD( INPUT dcName ).

/* Create a new row */
row1 = dataTable1:NewRow( ).

/* Add data to row. */
row1:Item["CustNum"] = 5.
row1:Item["Name"]    = "Mr Jones".

/* Process a value from the row. Without UNBOX, this does not compile.*/
iVal = UNBOX( row1:Item["CustNum"] ) MODULO 2. /* 1 = an odd value */
```

Notes

- You must use the UNBOX function if you want to reference an appropriate `System.Object` property or method return value in an ABL primitive expression, such as when performing arithmetic operations together with compatible ABL variables, fields, or literal values. Invoke the UNBOX function directly in the expression, passing it the `System.Object` as input, where you would otherwise reference the `System.Object` itself.
- For any direct assignment of a .NET object or object array to a compatible ABL primitive value or array, use of the UNBOX function is optional, because ABL automatically unboxes the underlying .NET object or array object type to its matching ABL primitive or array type.
- If you have a variable or field defined as a compatible ABL array type that you provide as an argument to an OUTPUT parameter of a **.NET** method defined as a .NET array object, ABL automatically unboxes the .NET array object into the ABL array argument. This automatic unboxing does **not** occur for an ABL method, procedure, or user-defined function passing the same parameters. In this case, you can use a direct assignment from a compatible .NET array object argument or use the UNBOX function to explicitly do the necessary conversion.

See also

BOX function, Data types

UNDERLINE statement

Underlines a field or variable, using the next display line for the underline.

Note: Does not apply to SpeedScript programming.

Syntax

```
UNDERLINE [ STREAM stream | STREAM-HANDLE handle ]
          field ... [ frame-phrase ]
```

`STREAM` *`stream`*

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream. See the DEFINE STREAM statement reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams.

`STREAM-HANDLE` *`handle`*

Specifies the handle to a stream. If *`handle`* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

`field`

Represents the name of the field or variable you want to underline.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

Example

This procedure produces a report of Customer records, categorized by State. When the last Customer for a certain state has been displayed (determined by the LAST-OF function), the UNDERLINE statement underlines the `State` field.

r-under1.p

```
FOR EACH Customer NO-LOCK BREAK BY Customer.State WITH USE-TEXT:
  DISPLAY Customer.State Customer.CustNum Customer.Name.
  IF LAST-OF(Customer.State) THEN UNDERLINE Customer.State.
END.
```

Notes

- Use The UNDERLINE statement to highlight fields or to underline accumulated values that you calculated using functions other than the automatic aggregate functions supplied with ABL.
- When determining the position within a DOWN frame, the DOWN statement and the UP statement count the line used by an underline.
- Even if the layout of a DOWN frame takes multiple screen lines, the underline takes just one line on the screen.
- For a 1 DOWN frame or single frame, the UNDERLINE does not appear. Instead, the AVM clears the frame.

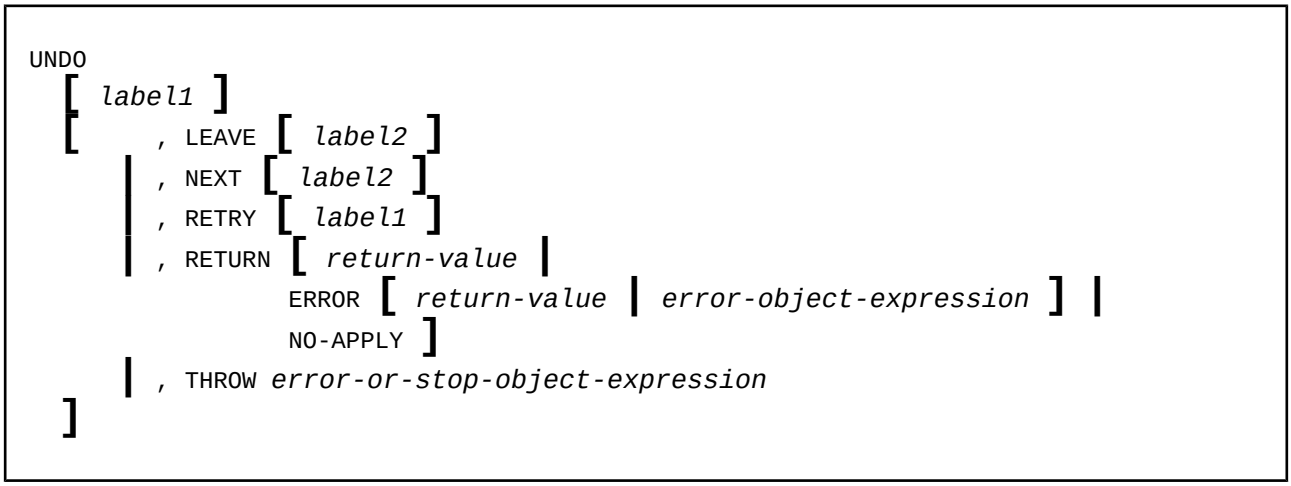
See also

DEFINE STREAM statement, Frame phrase, Stream object handle

UNDO statement

Backs out all modifications to fields and variables made during the current iteration of a block, and indicates what action to take next.

Syntax



label1

The name of the block whose processing you want to undo. If you do not name a block with *label1*, the AVM undoes the processing of the closest transaction or subtransaction block. In determining the closest transaction or subtransaction block, the AVM disregards DO ON ENDKEY blocks that do not have the ON ERROR or TRANSACTION option.

LEAVE *label2*

Indicates that after undoing the processing of a block, the AVM leaves the block you name with *label2*. If you do not name a block, the AVM leaves the block that was undone. After leaving a block, the AVM continues on with any remaining processing in a routine.

NEXT *label2*

Indicates that after undoing the processing of a block, the AVM does the next iteration of the block you name with *label2.* If you do not name a block, the AVM does the next iteration of the block that was undone.

RETRY *label1*

Indicates that after undoing the processing of a block, the AVM repeats the same iteration of the block you name with *label1*.

Because RETRY in a block without user input results in an infinite loop, the AVM automatically checks for this possibility and converts a RETRY block into a LEAVE block, or a NEXT block if it is an iterating block. This behavior is often referred to as infinite loop protection.

RETURN ...

Returns to the calling routine, if there is one. The following table describes various RETURN options:

Option	Description

return-value	In procedures and VOID methods, this must be a CHARACTER string. The caller can use the RETURN-VALUE function to read the returned value. For user-defined functions, non-VOID methods and property getters, the value must match the specified return type.
ERROR	Undoes the current subtransaction, and raises ERROR in the caller. You cannot specify ERROR within a user-interface trigger block or a destructor. For user-defined functions see note below.
ERROR *return-value*	Undoes the current subtransaction, and raises ERROR in the caller. The CHARACTER string you provide is available to the caller in the RETURN-VALUE function. The AVM also creates an AppError object and stores the *return-value* in the ReturnValue property. For user-defined functions see note below.
ERROR *error-object-expression*	Undoes the current subtransaction, and raises ERROR in the caller. The specified error object instance is thrown to the caller. For user-defined functions see note below.
NO-APPLY	In a user-interface trigger, prevents the AVM from performing the default behavior for the trigger event. Otherwise, the option is ignored.

Note: Using RETURN ERROR in a user-defined function sets the target variable of the function to the Unknown value (?) instead of raising ERROR in the caller. See *ABL Error Handling* for more detail.

THROW *error-or-stop-object-expression*

The THROW directive stops the execution of the current block of code, or the current iteration of an iterating block, and raises the ERROR or STOP condition specified in *error-or-stop-object-expression*. The value of *error-or-stop-object-expression* must be an error or stop object type.

UNDO, THROW has the following restrictions:

- UNDO, THROW is not allowed in a CATCH block associated with the main block of an object destructor method. You cannot raise or return an ERROR or STOP condition from a destructor. To do so results in a compile-time error. You can use UNDO, THROW within the code of the destructor itself. In this case, the statement raises the condition in the destructor block and is caught by the ON ERROR or ON STOP directive of the destructor block (which can only be UNDO, LEAVE).
- The UNDO, THROW statement can itself raise an ERROR condition, or THROW a Progress.Lang.SysError object if it fails. For example, this would happen if the object reference used is not initialized. In this case, the SysError is trapped by the same block that would have trapped the successfully thrown error or stop object.

Example

This example shows how the UNDO, THROW statement specifies and populates an error object, and how the CATCH block handles it:

r-undothrow1.p

```
FIND Customer 1000 NO-ERROR.
/* Raises error on current block (main block of .p); execution goes to CATCH
   below */
IF ERROR-STATUS:ERROR THEN
  UNDO, THROW NEW Progress.Lang.AppError("Can't find this customer", 550).

MESSAGE Customer.CustNum. /* This code does not execute if FIND fails */

/* This CATCH is on the main block of r-undothrow1.p */
CATCH eAppError AS Progress.Lang.AppError:
  MESSAGE eAppError:GetMessage(1) eAppError:GetMessageNum(1).
END CATCH.
```

Notes

- You can also specify UNDO processing for a block by using the ON ERROR, ON STOP, and ON ENDKEY phrases with a block statement.
- Class-based error or stop objects can be thrown from an OpenEdge application server and handled by a CATCH block on an ABL client. To be throwable from an OpenEdge application server to an ABL client, any user-defined error classes must be defined as SERIALIZABLE on both the server and client sides. In addition, note that any STOP condition messages are written to the server log file, even if a stop object with the messages is caught on the client. For the full list of restrictions on class-based objects that are passed between server and client, see the Parameter passing syntax entry.
- For more information on the UNDO statement, see *ABL Essentials*. For more information on error and stop handling in general, see *ABL Error Handling*.

See also

ON ENDKEY phrase, ON ERROR phrase, RETRY function, RETURN statement, ROUTINE-LEVEL ON ERROR UNDO, THROW statement

---

UNIX statement
(UNIX only)

Runs a program, UNIX command, or UNIX script, or starts a UNIX interactive shell to allow interactive processing of UNIX commands.

Syntax

```
UNIX
  [ SILENT ]
  [ command-token | VALUE ( expression ) ] ...
```

SILENT

After processing a UNIX statement, the AVM shell pauses and prompts you to press **SPACEBAR** to continue. You can use the SILENT option to eliminate this pause. Use this option only if you are sure that the UNIX program, command, or batch file does not generate any output to the screen.

command-token | VALUE (*expression*)

One or more command (*command-token*) words and symbols that you want to pass the UNIX operating system to execute. The VALUE option generates the command tokens included in *expression*, a character string expression. The specified combination of *command-token* and VALUE (*expression*) options can form any legal combination of commands and command options permitted by UNIX, including programs, built-in commands, and scripts. If you do not use any of these options, the UNIX statement invokes the UNIX shell and remains there until you press **CTRL+D** or the EOF character set by the UNIX `stty` command.

Examples

On UNIX, procedure `r-unix.p` starts a shell and in it runs the UNIX "ls" command. In Windows, this procedure starts a command processor and runs the DOS "dir" command in it:

r-unix.p

```
IF OPSYS = "UNIX" THEN UNIX ls.
ELSE IF OPSYS = "WIN32" THEN DOS dir.
ELSE DISPLAY OPSYS "is an unsupported operating system".
```

In `r-unx.p`, if you type an `L`, the AVM runs the DOS `dir` command or the UNIX `ls` command. If you enter a procedure name that is stored in the proc variable, the RUN statement then runs the procedure.

r-unx.p

```
DEFINE VARIABLE proc AS CHARACTER NO-UNDO FORMAT "x(40)".

REPEAT:
  DISPLAY "Enter L to list your files"
    WITH ROW 5 CENTERED FRAME a.
  SET proc LABEL "Enter a valid Procedure Name to run"
    WITH ROW 9 CENTERED FRAME b.
  IF proc = "L" THEN
    IF OPSYS = "UNIX" THEN UNIX ls.
    ELSE IF OPSYS = "WIN32" then DOS dir.
    ELSE display "Operating system" OPSYS "is not supported".
  ELSE DO:
    HIDE FRAME a.
    HIDE FRAME b.
    RUN VALUE(proc).
  END.
END.
```

Notes

- If you are using Windows and you use the UNIX statement in a procedure, that procedure will compile. The procedure will run as long as flow of control does not pass through the UNIX statement.
- This command does not exit to UNIX and return. It creates a shell within ABL to execute the command. Thus, you cannot use the UNIX statement as a substitute for the QUIT statement.

- When you use the UNIX cp command as an ABL statement, ABL assumes that a period (.) indicates the end of the statement. This causes the cp command to display a message stating that it requires two arguments. For example, ABL uses the period as the end of the statement indicator:

```
UNIX cp usr/myfile.
```

  To use the period as part of a UNIX command, enclose the command in quotation marks. For example:

```
UNIX "cp usr/myfile."
```

See also

DOS statement, OPSYS function, OS-COMMAND statement

UNLOAD statement
(Windows only)

Unloads a set of environment specifications from the current environment, which might be the registry or an initialization file.

Note: Does not apply to SpeedScript programming.

Syntax

```
UNLOAD environment [ NO-ERROR ]
```

environment

A character expression that evaluates to the name of an environment that a prior LOAD statement specified.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Notes

- An application cannot UNLOAD a set of environment specifications until it terminates all windows that use those specifications.
- If you UNLOAD the current environment, the default environment becomes the current environment. To define a new current environment, use the USE statement.
- Use the UNLOAD statement to clean up memory in applications, such as the AppBuilder, that build and run other applications.

See also

LOAD statement, NO-ERROR option, USE statement

UNSUBSCRIBE statement

Cancels a subscription to an ABL named event. Specifically, the UNSUBSCRIBE statement cancels one or more subscriptions to one or more named events.

Note: ABL named events are completely different from the key function, mouse, widget, and direct manipulation events, which are described in the Handle-based Object Events Reference. They are also different from the class events described in the Class Events Reference.

Syntax

```
UNSUBSCRIBE [ PROCEDURE subscriber-handle ]
  [ TO ] { event-name | ALL } [ IN publisher-handle ]
```

`PROCEDURE` *`subscriber-handle`*

A procedure handle representing the subscriber to a named event.

The PROCEDURE option lets one procedure cancel a subscription on behalf of another. For example, if you want procedure A to cancel a subscription on behalf of procedure B, set *`subscriber-handle`* to the procedure handle of B.

If the PROCEDURE option does not appear, the AVM assumes that the subscriber is THIS-PROCEDURE, the procedure that contains the UNSUBSCRIBE statement.

`event-name`

A quoted string or character expression representing the name of a named event.

`ALL`

Cancels all subscriptions.

`IN` *`publisher-handle`*

A procedure handle representing the publisher of a named event.

If the IN option appears, the AVM cancels subscriptions to named events published by *`publisher-handle`*-specifically, either all subscriptions (if the ALL option appears), or only subscriptions to *`event-name`* (if *`event-name`* appears).

If the IN option does not appear, the AVM cancels subscriptions regardless of the publisher-specifically, either all subscriptions (if the ALL option appears), or only subscriptions to *`event-name`* (if *`event-name`* appears).

Example

For an example, see the reference entry for the PUBLISH statement.

Notes

- When the AVM executes an UNSUBSCRIBE statement, it cancels a subscription when it finds a match. A match means that the SUBSCRIBE and UNSUBSCRIBE event names match, and that one of the following is true:
  - The subscription was created using SUBSCRIBE IN, cancelled using UNSUBSCRIBE IN, and the publisher and subscriber handles in the SUBSCRIBE and UNSUBSCRIBE statements match.
  - The subscription was created using SUBSCRIBE ANYWHERE, and cancelled using UNSUBSCRIBE without the IN option.
- The AVM executes the UNSUBSCRIBE statement with an implicit NO-ERROR option. That is, if the AVM cannot find a match, it does not report an error. To find out what errors, if any, occurred, use the ERROR-STATUS system handle.
- If you create a subscription using SUBSCRIBE ANYWHERE, you cannot cancel the subscription using UNSUBSCRIBE IN.

See also

PUBLISH statement, PUBLISHED-EVENTS attribute, SUBSCRIBE statement, Unsubscribe( ) event method

UP statement

Positions the cursor on a new line in a down or multi-line frame.

When the block specifying the down frame iterates, the AVM automatically advances one frame line. Use the UP statement if you want to move to a different display line at any time.

For more information on down frames, see the DOWN option of the Frame phrase.

Note: Does not apply to SpeedScript programming.

Syntax

```
UP [ STREAM stream | STREAM-HANDLE handle ][ expression ] [ frame-phrase ]
```

`STREAM` *`stream`*

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream. See the DEFINE STREAM statement reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams.

`STREAM-HANDLE` *`handle`*

Specifies the handle to a stream. If *`handle`* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

expression

Represents the number of occurrences of data in the frame that you want to move up. UP is the same as UP 1, except that nothing happens until the next data handling statement affects the screen. Several UP statements in a row with no intervening displays are treated like a single UP 1. UP 0 does nothing. If *expression* is negative, the result is the same as a DOWN *expression*.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase,* see the Frame phrase reference entry.

Example

This procedure starts at the bottom of the screen and displays all the Customer database records. The default frame for the FOR EACH block is a down frame. The DISPLAY statement uses that frame. Therefore, the AVM automatically advances down the screen one line after each iteration. You must use an UP 2 rather than an UP 1 because there is an automatic DOWN 1 performed on the display frame at the end of each iteration of the FOR EACH block.

r-up.p

```
FOR EACH Customer:
  UP 2.
  DISPLAY Customer.CustNum Customer.Name Customer.City.
END.
```

Notes

- When a frame is a down frame, the AVM automatically advances to the next frame line on each iteration of the block that it is scoped to, whether or not you use the DOWN statement. If you do not want the AVM to do this automatic advancing, name the frame outside of the current block.
- When the AVM reaches the top frame line and then encounters an UP statement, it clears the frame and starts at the bottom line of the frame. However, if you use SCROLL, the AVM moves everything in the frame down one row.

See also

DEFINE STREAM statement, DOWN statement, Frame phrase, SCROLL statement, Stream object handle

UPDATE statement

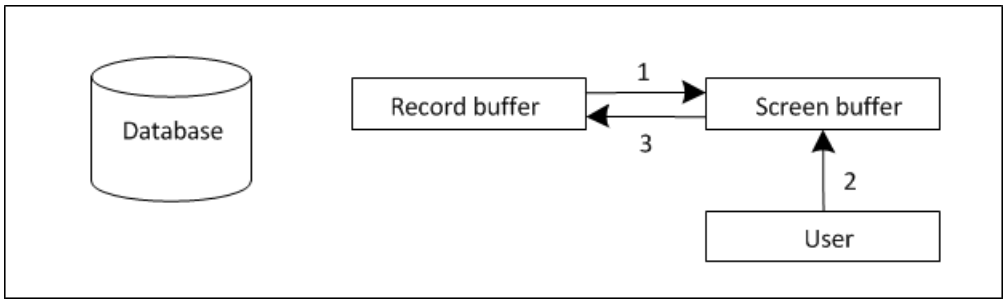
Displays fields or variables, requests input, and then puts the input data in both the screen buffer and in the specified fields or variables.

The UPDATE statement is a combination of the following statements:

- **DISPLAY** - Moves the values of fields or variables into the screen buffer and displays them (#1 in the figure)
- **PROMPT-FOR** - Prompts the user for data and puts that data into the screen buffer (#2 in the figure)
- **ASSIGN** - Moves data from the screen buffer to the record buffer (#3 in the figure)

Note: Does not apply to SpeedScript programming.

Data movement



Syntax

```
UPDATE
   [ UNLESS-HIDDEN ]
   [  field  [ format-phrase ] [ WHEN expression ]
      | TEXT ( field [ format-phrase ] ... )
      | field = expression
      | constant [ AT n  | TO n ]
      | ^
      | SPACE [ ( n ) ]
      | SKIP [ ( n ) ]
   ] ...
   [ GO-ON ( key-label ... ) ]
   [ frame-phrase ]
   [ editing-phrase ]
   [ NO-ERROR ]
```

```
UPDATE record
   [ EXCEPT field ... ]
   [frame-phrase ]
   [ NO-ERROR ]
```

`UNLESS-HIDDEN`

Restricts UPDATE to fields whose HIDDEN attribute is FALSE.

`field`

Represents the name of the field or variable whose value you want to display, change, and store in the screen and record buffers.

In array fields, array elements with constant subscripts are handled as any other field. Array fields with no subscripts are expanded as though you entered the implicit elements. See the DISPLAY statement reference entry for information on how array fields with expressions as subscripts are handled.

You can supply values for array elements in the UPDATE statement as follows:

```
UPDATE x[1] = "x".
```

This statement assigns the letter x to the first element of array x. If you do not include an array subscript, the AVM assigns the value to all elements of the array.

```
UPDATE X = "X".
```

This statement assigns the letter x to all elements of the array x.

format-phrase

Specifies one or more frame attributes for a field, variable, or expression. For more information on *format-phrase*, see the Format phrase reference entry.

WHEN *expression*

Updates the field only when *expression* has a value of TRUE. The *expression* is a field name, variable name, or expression whose value is logical.

TEXT

Defines a group of character fields or variables (including array elements) to use automatic word wrap. The TEXT option works only with character fields that are ABL default FILL-IN widgets (not specified with the FILL-IN NATIVE option). When you insert data in the middle of a TEXT field, the AVM wraps data that follows into the next TEXT field, if necessary. If you delete data from the middle of a TEXT field, the AVM wraps data that follows to the empty area. If you enter more characters than the format for the field allows, the AVM discards the extra characters. The character fields formats must be in the x(n) format. A blank in the first column of a line marks the beginning of a paragraph. Lines within a paragraph are treated as a group and will not wrap into other paragraphs.

The following table lists the keys you can use within a TEXT field and their actions.

Table 62: Key actions in a TEXT() field

Key	Action
APPEND-LINE	Combines the line the cursor is in with the next line.
BACK-TAB	Moves the cursor to the previous TEXT field.
BREAK-LINE	Breaks the current line into two lines beginning with the character the cursor is in.
BACKSPACE	Moves the cursor one position to the left and deletes the character at that position. If the cursor is at the beginning of a line, **BACKSPACE** moves the cursor to the end of the previous line.
CLEAR	Clears the current field and all fields in the TEXT group that follow.
DELETE-LINE	Deletes the line the cursor is in.

Key	Action
NEW-LINE	Inserts a blank line below the line the cursor is in.
RECALL	Clears fields in the TEXT group and returns initial data values for the group.
RETURN	In overstrike mode, moves to the next field in the TEXT group on the screen. In insert mode, the line breaks at the cursor and the cursor is positioned at the beginning of the new line.
TAB	Moves to the field after the TEXT group on the screen. If there is no other field, the cursor moves to the beginning of the TEXT group.

In this procedure, the s-com field is a TEXT field. Run the procedure and enter text in the field to see how the TEXT option works:

r-text.p

```
DEFINE VARIABLE s-com AS CHARACTER NO-UNDO FORMAT "x(40)" EXTENT 5.

FORM
  "Shipped   :" order.ship-date AT 13 SKIP
  "Misc Info :" order.instructions AT 13 SKIP(1)
  "Order Comments :" s-com AT 1
  WITH FRAME o-com CENTERED NO-LABELS TITLE "Shipping Information".

FOR EACH Customer NO-LOCK, EACH Order OF Customer:
  DISPLAY Customer.CustNum Customer.Name Order.OrderNum Order.OrderDate
    Order.PromiseDate WITH FRAME order-hdr CENTERED.
  UPDATE Order.ShipDate Order.Instructions TEXT(s-com) WITH FRAME o-com.
  s-com = "".
END.
```

field = expression

Indicates that the value of *field* is determined by evaluating the expression rather than having it entered on the screen or from a file. In effect, an assignment statement is embedded in the UPDATE statement.

constant AT *n*

Represents a constant value that you want to display in the frame. The *n* is the column in which you want to start the display.

constant TO *n*

Represents a constant value that you want to display in the frame. The *n* is the column in which you want to end the display.

^

Tells the AVM to ignore an input field when input is being read from a file. Also, the following statement reads a line from an input file and ignore that line:

```
UPDATE^
```

SPACE [(*n*)]

Identifies the number (*n*) of blank spaces to insert after the expression displays. The *n* can be 0. If the number of spaces you specify is more than the spaces left on the current line of the frame, the AVM starts a new line and discards any extra spaces. If you do not use this option or do not use *n*, the AVM inserts one space between items in the frame.

SKIP [(*n*)]

Identifies the number (*n*) of blank lines to insert after the expression is displayed. The *n* can be 0. If you do not use this option, the AVM does not skip a line between expressions unless they do not fit on one line. If you use the SKIP option, but do not specify *n* or if *n* is 0, the AVM starts a new line unless it is already at the beginning of a new line.

GO-ON (*keylabel . . .*)

Tells the AVM to take the GO action when the user presses any of the keys listed. You list keys in addition to keys that perform the GO action by default or because of ON statements. For example, if you want the AVM to execute the GO action when the user presses **F1**, use the statement GO-ON(**F1**). If you list more than one key, separate them with spaces, not commas.

Note that the GO-ON option is valid if you specify a list of fields in the UPDATE statement, but is invalid if you specify a record.

frame-phrase

Specifies the layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

editing-phrase

Supported only for backward compatibility.

Identifies processing to take place as each keystroke is entered. This is the syntax for *editing-phrase*:

```
[ LABEL : ] EDITING : statement . . . END
```

For more information on *editing-phrase*, see the EDITING phrase reference entry.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

record

Specifies the name of a record buffer. All of the fields in the record are processed as if you updated each of them individually.

To update a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

EXCEPT *field*

Affects all fields except those fields listed in the EXCEPT phrase; they are omitted from the update list.

Note: Compound assignment operators (+=, -=, *=, /=) can be used with the UPDATE statement. For more information, see += Addition assignment operator, -= Subtraction assignment operator, *= Multiplication assignment operator, and /= Division assignment operator.

Examples

The following procedure lets you update the Name, Address, City, State, and Country for each Customer record in the database:

r-updat.p

```
FOR EACH Customer:
  UPDATE Customer.Name Customer.Address Customer.City Customer.State
    Customer.Country.
END.
```

The r-updat2.p procedure reads each Customer record and lets you update the Name and CreditLimit fields. The VALIDATE option on the first UPDATE statement ensures that you enter a CreditLimit value that is less than 500000. The HELP option displays a message to that effect.

r-updat2.p

```
FOR EACH Customer:
  UPDATE Customer.Name
    Customer.CreditLimit VALIDATE(Customer.CreditLimit < 500000, "Too high")
    HELP "Enter CreditLimit < 500000".

  FOR EACH Order OF Customer:
    DISPLAY Order.OrderNum.
    UPDATE Order.PromiseDate Order.ShipDate VALIDATE(Order.ShipDate > TODAY,
      "Ship date must be later than today").
  END.
END.
```

The second FOR EACH block reads every Order belonging to the Customer, displays the OrderNum field, and lets you update the PromiseDate and ShipDate fields. The VALIDATE option ensures that you enter a ship date value that is after today's date.

This procedure requests a Customer number and then lets you update information for that Customer record. The frame phrase WITH 1 COLUMN 1 DOWN tells the AVM to display the fields in a single column on the screen (rather than in a row across the screen) and to display only one Customer record on the screen at a time.

r-updat3.p

```
REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum.
  UPDATE Customer.Name Customer.Address Customer.City Customer.State
    Customer.Country WITH 1 COLUMN 1 DOWN.
END.
```

Notes

- If any *field* is a field in a database record, the UPDATE statement upgrades the record lock condition to EXCLUSIVE-LOCK before updating the record.
- If any *field* is part of a record retrieved with a field list, the UPDATE statement rereads the complete record before updating it. If any *field* is not part of the field list (or related fields) fetched with the record, or if *record* includes such unfetched fields, the AVM raises the ERROR condition **before** the UPDATE statement accepts input. This is because the UPDATE attempts to display the fields before it rereads the record.
- If an error occurs during UPDATE statement input (for example, the user enters a duplicate index value for a unique index), the AVM retries the data entry part of the statement and does not do the error processing associated with the block that contains the statement.
- The UPDATE statement is **not** equivalent to a combination of the DISPLAY and SET statements.

```
REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum.
  UPDATE Customer.CreditLimit.
END.
```

The previous procedure is approximately equivalent to the following procedure:

```
REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum.
  DISPLAY Customer.CreditLimit.

  DO ON ERROR UNDO, RETRY:
    SET Customer.CreditLimit.
  END.
END.
```

If an error occurs during an UPDATE statement, the statement is retried until the error is corrected. If this happens during a SET statement, an entire block is retried.

- If you receive input from a device other than the terminal, and the number of characters read by the UPDATE statement for a particular field or variable exceeds the display format for that field or variable, the AVM returns an error. However, if you are setting a logical field that has a y/n format and the data file contains a value of YES or NO, the AVM converts that value to "y" or "n".
- If you use a single qualified identifier with the UPDATE statement, the compiler first interprets the reference as *dbname.tablename*. If the compiler cannot resolve the reference as *dbname.tablename*, it tries to resolve it as *tablename.fieldname*.
- When updating fields, you must use table names that are different from field names to avoid ambiguous references. See the Record phrase reference entry for more information.
- The UPDATE statement causes ASSIGN and WRITE events to occur and all related database ASSIGN and WRITE triggers to execute. The ASSIGN triggers execute before the WRITE triggers and after the field is actually updated. The WRITE triggers only execute if the ASSIGN triggers do not return an error. If an ASSIGN trigger fails, the database update is undone. This means that all database changes are backed out. If the UPDATE statement occurs within a transaction, any changes to variables, worktable fields, and temp-table fields are also undone unless the variable or field is defined with the NO-UNDO option. Likewise, if a WRITE trigger fails, the UPDATE statement is undone.

- When you execute UPDATE with a specific or implied GO-ON(*keylabel*) from a called program, the AVM generates an error message (4123). This is due to an incompatibility in focus. The workaround is to add a VIEW FRAME statement after the call to the subprocedure such that the VIEW FRAME is the first statement executed on return from the called procedure.
- In the context of the .NET blocking method, `System.Windows.Forms.Application:Run( )`, if you execute the UPDATE statement for a non-modal ABL window from within a user-defined function or non-VOID method, this statement raises the STOP condition. For more information on the .NET `Application:Run( )` method, see the reference entry for the WAIT-FOR statement (.NET and ABL).

See also

ASSIGN statement, DISPLAY statement, EDITING phrase, Format phrase, Frame phrase, NO-ERROR option, PROMPT-FOR statement

---

USE statement
(Windows only)

Specifies environment defaults that apply to subsequent windows that the application creates. The defaults might reside in the registry or in an initialization file. The defaults can involve colors, fonts, environment variables, etc. You must specify a default in a LOAD statement before you specify it in a USE statement.

Note: Does not apply to SpeedScript programming.

Syntax

```
USE environment [ NO-ERROR ]
```

environment

A CHARACTER expression that evaluates to the name of a current environment. If *environment* is non-null, it must have appeared in a prior LOAD statement. If *environment* is the null string (""), the default environment becomes the current environment.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

This procedure loads two files, env1.ini and env2.ini, each of which contains a font definition for font0. The program displays a character string in the ABL default window using the definition for font0 from env1.ini. It then creates a new window and displays the same character string using the definition for font0 from env2.ini. Note that the procedure creates the window after the USE statement.

r-use.p

```
DEFINE VARIABLE w1      AS CHARACTER NO-UNDO VIEW-AS TEXT FONT 0
  FORMAT "x(34)" INITIAL "This is font 0 in the first window".
DEFINE VARIABLE w2      AS CHARACTER NO-UNDO VIEW-AS TEXT FONT 0
  FORMAT "x(35)" INITIAL "This is font 0 in the second window".
DEFINE VARIABLE new_win AS HANDLE    NO-UNDO.

LOAD "env1".
LOAD "env2".

USE "env1".
DISPLAY w1 WITH NO-LABELS WITH FRAME a.
PAUSE.

USE "env2".
CREATE WINDOW new_win.
CURRENT-WINDOW = new_win.
DISPLAY w2 in WINDOW new_win WITH NO-LABELS WITH FRAME b.
PAUSE.

DELETE WIDGET new_win.
```

This procedure depends on the existence of files named env1.ini and env2.ini, each of which contains a font definition for font0. If you run this procedure in your environment, you must create these files.

Notes

- The USE statement does not change the DefaultFont or the DefaultFixedFont settings. It uses only the information in the [fonts] and [colors] sections of the new current environment's initialization file.
- Use this statement with applications (such as the AppBuilder) that build and run other applications using a unique set of environment specifications.
- An application must use this statement after the LOAD statement and before a new window is created to make the loaded set of environment specifications apply to the new window.
- Subsequent PUT-KEY-VALUE and GET-KEY-VALUE statements apply to the environment made available by the USE statement.

See also

GET-KEY-VALUE statement, LOAD statement, NO-ERROR option, PUT-KEY-VALUE statement

USERID function

Returns a character string representing the user ID for the specified database connection identity.

Syntax

```
USERID [ ( logical-dbname ) ]
```

logical-dbname

The logical name of the database from whose connection identity you want to retrieve the user ID. The logical database name must be a character string enclosed in quotes, or a character expression. If you do not specify this argument, the compiler inserts the name of the database that is connected when the procedure is compiled. If you omit this argument and more than one database is connected, ABL raises an error.

Example

This one-line procedure displays the current user ID for the database with the DICTDB alias:

r-userid.p

```
DISPLAY USERID("DICTDB") LABEL "You are logged in as" WITH SIDE-LABELS.
```

Notes

- The user ID returned by this function can be set for a database connection identity that is specified on the command line or when executing the CONNECT statement, the SECURITY-POLICY:SET-CLIENT( ) method, the SET-DB-CLIENT function, or the SETUSERID function.
- For a single-tenant database connection, the user ID returned by the USERID function is non-qualified; for a multi-tenant database connection, it is fully qualified, as shown in the following table.

  Table 63: Values returned from the USERID function

This value is returned...	For...
""	A non-qualified user ID for the blank user name in the blank domain
"*user-name*"	A non-qualified user ID for a specific user name in the blank domain
"*@domain-name*"	A fully qualified user ID for the blank user name in a specific domain
"*user-name@domain-name*"	A fully qualified user ID for a specific user in a specific domain

  The user ID returned by this function is always consistent with the user ID used in database identity audit trails, as well as for applying ABL table and field permissions and ABL Security Administrator rights.

- When using the USERID function, ABL returns a compiler error under the following conditions:
  - There is no database connected
  - You omit the *logical-dbname* argument and more than one database is currently connected
- When specifying the *logical-dbname* argument, you must provide the name of the logical database, not the physical database.
- After ABL starts running, you can use the SET-CLIENT( ) method, the SET-DB-CLIENT function, or the SETUSERID function to change the current user ID associated with an existing database connection.
- ABL user IDs in `_User` table accounts are case insensitive.

- See *Learn about Identity Management*, *OpenEdge Programming Interfaces*, and *Manage the OpenEdge Database* for more information on user identity.

See also

CAN-DO function, CONNECT statement, QUALIFIED-USER-ID attribute, SET-CLIENT( ) method, SET-DB-CLIENT function, SETUSERID function, USER-ID attribute

USING statement

Allows you to reference a single specified ABL or .NET object type, or reference all the types in an ABL package or .NET namespace, using their unqualified type names.

Syntax

```
USING
  { object-type-name | { package-name | namespace } .* }
  [ FROM { ASSEMBLY | PROPATH } ]
```

object-type-name

Specifies the qualified type name of a single ABL class or interface, or of a single .NET class, structure, interface, or enumeration. This type name must be the fully qualified type name (including the package or namespace) of an ABL or .NET class or interface type (as described in the Type-name syntax reference entry).

Use this option to identify a single ABL class or interface defined in a package, or a single .NET class, structure, interface, or enumeration defined in a namespace, that you want to reference by its unqualified type name.

The value of *object-type-name* is restricted to alphanumeric characters plus the symbols #, $, %, and _.

{ *package-name* **|** *namespace* **}** .*

Specifies an ABL *package-name* that includes ABL classes or interfaces that you want to reference, or a .NET *namespace* that includes .NET classes (including structures and enumerations) or interfaces that you want to reference, where an unqualified class, interface, structure, or enumeration name can replace '*' to form a valid fully qualified type name.

Use this option to identify all the ABL classes or interfaces defined in a package, or all the .NET classes, structures, interfaces, or enumerations defined in a namespace, for reference by their unqualified type names.

This expression is restricted to alphanumeric characters plus the symbols #, $, %, and _.

[FROM { ASSEMBLY | PROPATH }]

Optionally identifies where to locate a given type definition whose type is being referenced using this statement. This identification permits the ABL compiler to focus its search for the type definition depending on whether the USING statement specifies an ABL package-based type or a .NET namespace-based type. When you specify the FROM ASSEMBLY option, ABL assumes that the type is a .NET type and uses the list of .NET assemblies defined in the assembly identification file to search for the type definition. When you specify the FROM PROPATH option, ABL assumes that the type is an ABL type and uses PROPATH to search for the type definition. If you do not specify a FROM option, the search for a type definition first uses PROPATH, then uses the specified list of .NET assemblies. In most cases, the use of an appropriate FROM option improves compile-time performance.

Caution: Avoid using ABL and .NET class types with the same name. This can cause unpredictable behavior, regardless of whether you specify FROM PROPATH or FROM ASSEMBLY.

Example

You can use either one of the following code fragments to reference the class `Acme.BusObjs.Customer` using its unqualified or fully qualified class name:

```
USING Acme.BusObjs.*.

DEFINE VARIABLE CustObj  AS CLASS Customer.
DEFINE VARIABLE CustObj2 AS CLASS Acme.BusObjs.Customer.
```

```
USING Acme.BusObjs.Customer.

DEFINE VARIABLE CustObj  AS CLASS Customer.
DEFINE VARIABLE CustObj2 AS CLASS Acme.BusObjs.Customer.
```

The `ControlCollection` class is an inner class of the .NET class `System.Windows.Forms.Control`. Therefore, you can reference these class types by their unqualified type names like this:

```
USING System.Windows.Forms.* FROM ASSEMBLY.

DEFINE VARIABLE rControl    AS CLASS Control.
DEFINE VARIABLE rCollection AS CLASS Control+ControlCollection.
```

Notes

- When a type name is referenced at compile time, ABL uses the specified USING statements to help validate that the type exists in a specified package or loaded assembly.
- This statement must appear before any other compilable statement (except other USING statements or the ROUTINE-LEVEL ON ERROR UNDO, THROW statement) in a procedure (`.p` or `.w`) or class definition (`.cls`) file. The scope of the statement is the file in which it appears.
- This statement has no effect on the *`object-type-name`* that defines the name of a user-defined type in a CLASS or INTERFACE statement. In a CLASS or INTERFACE statement, you must always specify the user-defined type that it defines using the fully qualified *`object-type-name`*.

- For both ABL and .NET object types, you can only reference class names for classes, interface names for interfaces (and so on) that are defined in the specified ABL package or .NET namespace. For example, you cannot reference partially qualified class names for classes in packages that are further defined under the specified package. To reference the class names of classes in other packages, even those that appear to be subpackages, you must specify an additional USING statement for each package. For example, given the following class file:

```
C:\Classes\Inventory\Shipping.cls
```

  If you specify the USING statement in this code fragment, the class references in the following statements are invalid:

```
USING Classes.*.

DEFINE VARIABLE clRef AS Inventory.Shipping. /* Invalid */
clRef = NEW Inventory.Shipping().            /* Invalid */
```

  The following code fragment is correct:

```
USING Classes.Inventory.*.

DEFINE VARIABLE clRef AS Shipping.
clRef = NEW Shipping().
```

- You can also reference the fully qualified type names of any object types identified by this statement. You must sometimes do this to avoid ambiguous class or interface name references.
- When you reference a constructed .NET generic type name with the presence of appropriate USING statements, you can also specify the type parameters in its name, as well as the generic type name, itself, using unqualified type name references. This also works to resolve unqualified type-name references to the type parameters specified in the generic *object-type-name* of successive USING statements. In addition, ABL resolves unqualified type-name references to all the constructed generic types that can be defined for a given *namespace* specified in a USING statement. For example:

```
/* 1 */ USING System.Windows.Forms.*.

/* Button is resolved by the first USING statement */
/* 2 */ USING "System.Collections.Generic.List<Button>".

/* 3 */ USING System.Collections.ObjectModel.*.

/* List and Button are resolved by the first two USING statements */
DEFINE VARIABLE ButtonList AS CLASS "List<Button>" NO-UNDO.

/* "Collection<INTEGER>" is resolved by the third USING statement */
DEFINE VARIABLE intColl AS CLASS "Collection<INTEGER>" NO-UNDO.

/* However, this line does not compile */
DEFINE VARIABLE stringList AS CLASS "List<CHARACTER>" NO-UNDO.
```

Note that unlike the previous DEFINE statement, the final DEFINE statement does not compile, because there is no USING statement specified to resolve references to all possible constructions of the defined open generic type, in this case, `System.Collections.Generic.List<T>`. Also in this example, `Button` is a class in the `System.Windows.Forms` namespace and `Collection<T>` is a generic class in the `System.Collections.ObjectModel` namespace. For more information on referencing .NET generic types in ABL, see the Data types reference entry.

- ABL does not validate the definition of type names, packages, or namespaces during USING statement compilation. When ABL later encounters an unqualified object type name following the compilation of all USING statements, it validates this type name against each available *object-type-name*, *package-name*, and *namespace* specified by these USING statements to identify and verify the type definition.
- During procedure or class definition file compilation, ABL resolves all unqualified type name references according to the following algorithm:

  1. ABL examines the USING statements in order and, for each statement that specifies a *object-type-name*, compares the unqualified name with the last node of the *object-type-name*. If the names are equal, ABL considers this a match and continues compilation with the current USING statement according to its specified FROM option:
     - If FROM ASSEMBLY is specified, ABL searches for *object-type-name* in the loaded assemblies.
     - If FROM PROPATH is specified, ABL checks if *object-type-name* specifies a built-in class or interface type, and if not, ABL searches for a corresponding class file according to the *object-type-name* package relative to PROPATH.
     - If no FROM option is specified, ABL checks if *object-type-name* specifies built-in class or interface type, and if not, ABL first searches for a corresponding class file according to the *object-type-name* package relative to PROPATH, then searches for *object-type-name* in the loaded assemblies.
  2. If ABL has not matched the unqualified type name with a *object-type-name* specified in a USING statement, ABL again examines the USING statements in order. For each USING statement that contains a *package-name*.* or *namespace*.*, ABL replaces the "*" with the unqualified name, creating a *object-type-name*. ABL then continues compilation with the current USING statement according to its FROM option:
     - If FROM ASSEMBLY is specified, ABL searches for *object-type-name* in the loaded assemblies.
     - If FROM PROPATH is specified, ABL checks if *object-type-name* specifies a built-in class or interface type, and if not, ABL searches for a corresponding class file according to the *object-type-name* package relative to PROPATH.
     - If no FROM option is specified, ABL checks if *object-type-name* specifies built-in class or interface type, and if not, ABL first searches for a corresponding class file according to the *object-type-name* package relative to PROPATH, then searches for *object-type-name* in the loaded assemblies.
  3. If ABL does not identify a qualified type name from any *object-type-name* derived from a USING statement, it searches for the unqualified type name reference as a complete ABL type name (defined without a *package-name*). Thus, ABL searches for an ABL class file with a filename equal to the unqualified type name directly on PROPATH.

  If none of these searches identify a valid type definition, ABL generates a compiler error that the type cannot be found.

See also

CLASS statement, INTERFACE statement, Type-name syntax

VALID-EVENT function

Verifies whether a specified event is valid for a specified widget. For each type of widget, only certain events are valid. The function returns a value (TRUE/FALSE).

Note: Does not apply to SpeedScript programming.

Syntax

```
VALID-EVENT ( handle , event-name [ , platform ] )
```

handle

An expression that produces a value of type HANDLE. The value must be the handle of a valid widget.

event-name

A character-string expression that evaluates to the name of an event.

platform

A character-string expression that evaluates to the name of a platform type: GUI or TTY.

See also

LAST-EVENT system handle, LIST-EVENTS function, LIST-QUERY-ATTRS function, LIST-SET-ATTRS function, LIST-WIDGETS function

VALID-HANDLE function

Verifies that a handle is valid.

Note: Does not apply to SpeedScript programming.

Syntax

```
VALID-HANDLE ( handle )
```

handle

An expression that evaluates to a value of type HANDLE. If the handle represents an object that is currently valid, VALID-HANDLE returns TRUE. If the handle is no longer valid (if, for example, some procedure deleted the object), the function returns FALSE.

Example

In the following example, the user creates a window dynamically. The WINDOW-CLOSE trigger uses the VALID-HANDLE function to determine whether the window has been created.

r-valhnd.p

```
DEFINE VARIABLE mywin AS HANDLE NO-UNDO.

DEFINE BUTTON mkwin LABEL "New Window".

ENABLE mkwin.

ON CHOOSE OF mkwin DO:
  CREATE WINDOW mywin ASSIGN
    VISIBLE          = TRUE
    TITLE            = "Second Window"
    MAX-WIDTH-CHARS  = 40
    MAX-HEIGHT-CHARS = 10.
  SELF:SENSITIVE   = FALSE.
END.

ON WINDOW-CLOSE OF DEFAULT-WINDOW DO:
  IF VALID-HANDLE(mywin) THEN
    DELETE WIDGET mywin.
END.

WAIT-FOR WINDOW-CLOSE OF DEFAULT-WINDOW.
```

In the example, the VALID-HANDLE function returns a TRUE value only if the window has been created (that is, `mywin` does not have the Unknown value (?)) and the window has not been deleted. Therefore, the DELETE WIDGET statement executes only if `mywin` is a valid handle.

Notes

- A handle becomes invalid if the associated widget or procedure is deleted or is out of scope.
- This function is useful when walking through a list of widgets or persistent procedures using the PREV-SIBLING or NEXT-SIBLING attributes.

  VALID-HANDLE(*handle*:PREV-SIBLING) is FALSE when you reach the first handle in the list. VALID-HANDLE(*handle*:NEXT-SIBLING) is FALSE when you reach the last handle in the list.
- If a handle is valid, it can still point to an obsolete object. That is, ABL can recycle a previously used handle value to reference a new object instance, leaving previously set handles with the same value pointing to older objects. So you can recognize that seemingly identical handles are actually handle copies that point to different objects, ABL supports the UNIQUE-ID attribute on some types of object handles. For more information, see the UNIQUE-ID attribute reference entry.
- The VALID-HANDLE function supports handles to application servers, proxy procedures, remote persistent procedures, and COM objects. For more information on COM objects, see *OpenEdge Programming Interfaces*.

See also

CREATE SERVER statement, HANDLE function

VALID-OBJECT function

Verifies that an object reference points to a valid ABL or .NET class-based object instance or an ABL handle-based object. If the object reference represents an object that is currently valid, the function returns TRUE. If the object reference is no longer valid (for example, it was garbage collected or explicitly deleted), the function returns FALSE.

Syntax

```
VALID-OBJECT ( { handle | object-reference } )
```

handle

A handle to an ABL handle-based object. The handle argument must be a variable of type HANDLE and must contain a valid handle. When used with a handle-based object, VALID-OBJECT works in the same way as the VALID-HANDLE function.

object-reference

An object reference defined for an ABL or .NET object type. For more information, see the Class-based object reference reference entry.

See also

DELETE OBJECT statement, VALID-HANDLE function

VALIDATE statement

Verifies that a record complies with mandatory field and unique index definitions.

Syntax

```
VALIDATE record [ NO-ERROR ]
```

record

The name of the record you want to validate.

To validate a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the *record* definition in the Record phrase reference entry for more information.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

Example

This procedure prompts for an item number. If an Item with that number is not available, the procedure creates a new Item record and lets you supply some Item information. The VALIDATE statement checks the data you enter against the index and mandatory field criteria for the Item record.

r-valid.p

```
REPEAT FOR Item:
  PROMPT-FOR Item.ItemNum.
  FIND Item USING Item.ItemNum NO-ERROR.
  IF NOT AVAILABLE Item THEN DO:
    CREATE iIem.
    ASSIGN Item.ItemNum.
    UPDATE Item.ItemName Item.Price.
    VALIDATE Item.
  END.
  ELSE
    DISPLAY Item.ItemName Item.Price.
END.
```

Notes

- Because validation is done automatically, you rarely have to use the VALIDATE statement. The AVM automatically validates a record when a record in the record buffer is replaced by another, a record's scope iterates or ends, the innermost iterating subtransaction block that creates a record iterates, or a transaction ends.
- The AVM automatically validates mandatory fields when those fields are modified.
- If the validation fails on a newly-created record, VALIDATE raises the ERROR condition.
- The AVM performs validation when it leaves a field.
- For complex validations, it might be easier to use the IF...THEN...ELSE statement instead of the VALIDATE statement.
- You cannot use the VALIDATE statement to test fields that are referenced in SQL statements, since validation is not performed for these fields.
- If a field or table has been modified, the VALIDATE statement causes WRITE events and all related WRITE triggers to execute.

See also

IF...THEN...ELSE statement, NO-ERROR option

VAR statement

Defines a variable. The VAR statement is a short-hand notation for the DEFINE VARIABLE statement. Variables defined with the VAR statement are NO-UNDO, meaning that the value of the variable will not be restored if the variable changed during a transaction and the transaction is undone. If you wish to have the option to restore the variable to its prior value, use the DEFINE VARIABLE statement instead.

Syntax

```
VAR [ access-mode ] [ STATIC | SERIALIZABLE | NON-SERIALIZABLE ]
    type [ [ [ size ] ] ] | [ CLASS ] classname [ [ [ size ] ] ]
    variable [ = expression [, expression ]... ].
```

access-mode

Specifies an access mode (PRIVATE, PACKAGE-PRIVATE, PROTECTED, PACKAGE-PROTECTED, or PUBLIC) for the user-defined class variable. For more information, see Access modes in *Object-oriented ABL Developer Guided Journey*.

STATIC

Indicates that the user-defined class variable is scoped to the ABL session where it is referenced. ABL creates one copy of the specified class static variable on first reference to the class type, and ABL creates only one such copy for any number of instances of the class that you create. You can reference an accessible static variable data member in any piece of code

Without the STATIC option, the user-defined class variable is scoped to a single instance of the class where it is defined. ABL creates one copy of the specified instance variable for each such class instance that you create. You can reference any public instance variable in any procedure, or in any instance or static method defined inside or outside of the class where the instance variable is defined. Any static method can reference the public instance variable only using an object reference to a class instance that defines the variable as a data member. If the referencing static method is defined in the same class as the public instance variable, the class must instantiate itself in order to have access to an instance reference.

SERIALIZABLE | NON-SERIALIZABLE

Indicates whether the user-defined class variable will participate in serialization. SERIALIZABLE and NON-SERIALIZABLE options can only be used with instance (not STATIC) variables. For more information, see Serialization and deserialization in *Develop Object-oriented ABL Applications*.

type

Specifies a built-in data type for the variable you are defining. *Type* can be one of the following: CHAR, CHARACTER, COM-HANDLE, DATE, DATETIME, DATETIME-TZ, DECIMAL, HANDLE, INT, INTEGER, INT64, LOGICAL, LONGCHAR, MEMPTR, RAW, RECID, or ROWID. *Type* cannot be abbreviated.

Note: CHAR is a synonym for CHARACTER and INT is a synonym for INTEGER.

For more information on ABL data types, see Data types.

[*size*]

Indicates that the variable is an array. *size* is optional and specifies the number of elements in the array. If *size* is specified, it must be an integer constant. If *size* is not specified ([]), then the variable is considered an indeterminate array. For more information about arrays, see Work with one-dimensional arrays in *Basic ABL*.

variable

Specifies the name of the variable. Variable names must begin with a letter, but subsequent characters can include letters, numeric digits, periods(.), hyphens(-), and underscores(_). Variable names are case-insensitive.

expression

An Expression which evaluates to a data type that is consistent with the data type of the defined variable.

Examples

The following example code shows variable declarations using VAR.

```
/* Four character variables with default initial values */
VAR CHAR s1, s2, s3, s4.

/* Three integer variables. */
/* z's initial value is 3. x and y default to 0 */
VAR INT x, y, z = 3.

/* Three date variables */
VAR DATE d1, d2 = 1/1/2020, d3 = TODAY.

/* Date variables with the protected access mode */
VAR PROTECTED DATE d1, d2 = 1/1/2020.

/* Three arrays of size 3 */
/* x's third element defaults to 2 (the previous element) */
/* y's elements default to 0 */
VAR INT[3] x = [1, 2], y, z = [100, 200, 300].

/* Two indeterminate arrays. x has no size. y has a size of 3. */
VAR INT[] x, y = [1,2,3].

/* .NET generic object */
VAR "System.Collections.Generic.List<char>" cList.

/* Three object variables */
VAR mypackage.subdir.myclass myobj1, myobj2, myobj3.

/* Object variable with the optional CLASS keyword */
VAR CLASS mypackage.subdir.myclass myobj1.

/* Object array of size 2 */
VAR foo[2] classArray.

/* Instantiated object variables */
VAR myclass myobj = NEW myclass().
VAR myclass myobj = NEW myclass("Progress", 2020,?).

/* Multiple instantiated object variables */
VAR myclass myobj1 = NEW myclass("MA"),
            myobj2 = NEW myclass("VT"),
            myobj3 = NEW myclass("NH").

/* Instantiated base class type */
VAR baseclass myobj1 = NEW derivedAClass(), myobj2 = NEW derivedBClass("foo").

/* Instantiated interface type */
VAR IMaps myobj1 = NEW GoogleMap("CANADA"), myobj2 = NEW OpenStreetMap("USA").

/* Instantiated and initialized determinate array object variable */
VAR StateClass[2] objArrayA = [NEW StateClass(“MA”), NEW StateClass(“NH”)].

/* Instantiated and initialized indeterminate array object variable */
VAR StateClass[] objArrayB = [NEW StateClass(“MA”), NEW StateClass(“NH”)].

/* Integer variables initialized using expressions */
VAR INT x = a + b , y = a - b, z = x - y.

/* Indeterminate integer array initialized using expressions */
VAR INT[ ] x = [funct( ), a + b].
```

Notes

- VAR may only be used at the beginning of an ABL routine, before any executable statements in the block.

If in a procedure, the VAR statement must follow these statements:

- `USING`...
- `BLOCK-LEVEL ON ERROR UNDO, THROW.`
- `ROUTINE-LEVEL ON ERROR UNDO, THROW.`

If in a class, the VAR statement:

- Must come after the `CLASS` statement.
- Can be in any order within the class, like other class members.
- Must be at the top of the block within a method or property body.

- VAR does not allow abbreviated data types. For example, when specifying a decimal variable, you cannot write `VAR DEC`. You must write `VAR DECIMAL`. In Release 12.3, CHAR and INT are actual keywords and not abbreviations. CHAR is a synonym for CHARACTER and INT is a synonym for INTEGER. Therefore you can write `VAR CHAR` or `VAR INT`.
- Other options allowed with the `DEFINE VARIABLE` statement, such as `FORMAT` or `LABEL`, cannot be used with VAR. If you need those other options, use the `DEFINE VARIABLE` statement instead.
- Uninitialized elements in a determinate array object variable are set to the Unknown (?) value, unlike scalar array variables where uninitialized elements are set to the last initialized value, if any.

See also

DEFINE VARIABLE statement, Data types, EXTENT statement, NEW function (classes)

VIEW statement

Displays a widget (sets its VISIBLE attribute to TRUE).

Note: Does not apply to SpeedScript programming.

Syntax

```
VIEW
   [ STREAM stream | STREAM-HANDLE handle ]
   [ widget-phrase ]
   [ IN WINDOW window ]
```

`STREAM` *`stream`*

Specifies the name of a stream. If you do not name a stream, the AVM uses the unnamed stream.

`STREAM-HANDLE` *`handle`*

Specifies the handle to a stream. If *`handle`* it is not a valid handle to a stream, the AVM generates a run-time error. Note that stream handles are not valid for the unnamed streams. See the topics on alternate I/O sources in *OpenEdge Programming Interfaces* for more information on streams and stream handles.

widget-phrase

Specifies the widget you want to view. You can view windows, frames, and field-level widgets. You cannot view menus. If you do not use this option, VIEW sets the VISIBLE attribute for the default frame for the current block.

IN WINDOW *window*

Specifies the window in which to view the widget.

Example

The r-view2.p procedure displays information on a sales representative and then displays all the Customers belonging to that sales representative. Each new sales representative is displayed on a new page. In addition, if the information for a sales representative takes up more than one page, a separate FORM statement describes a continuation header for that sales representative. The VIEW statement for the PAGE-TOP frame hdr2, activates the header for subsequent page breaks.

r-view2.p

```
OUTPUT TO slsrep PAGED PAGE-SIZE 10.

FOR EACH SalesRep NO-LOCK:
  PAGE.
  FORM HEADER "Sales rep report" "Page" AT 60 PAGE-NUMBER FORMAT ">>>9".

  DISPLAY SKIP(1) SalesRep.SalesRep SalesRep.RepName SalesRep.Region
    WITH NO-LABELS.

  FORM HEADER "Sales rep report" SalesRep.SalesRep "(continued)"
    "Page" AT 60 PAGE-NUMBER FORMAT ">>>9" SKIP(1)
    WITH FRAME hdr2 PAGE-TOP.
  VIEW FRAME hdr2.

  FOR EACH Customer OF SalesRep NO-LOCK:
    DISPLAY Customer.CustNum Customer.Name Customer.Address Customer.City
      Customer.State.
  END.
END.
```

Notes

- If the widget is already visible, the VIEW statement has no effect.
- Viewing a widget does not, by itself, show any of its data. To view data in a widget, you must use a data display statement (such as DISPLAY) or assign the data directly to the widget's SCREEN-VALUE attribute.
- When you view a window, its frames and their descendant widgets are not displayed, unless you explicitly view or display them.
- When you view a widget, the AVM displays that widget unless its parent window or an ancestor window has its HIDDEN attribute set to TRUE.
- When you view a widget that has its HIDDEN attribute set to TRUE, the AVM sets the widget's HIDDEN attribute to FALSE.
- When you view a widget contained by a window that is invisible (VISIBLE attribute is FALSE), that widget and the containing window is displayed unless the containing window's HIDDEN attribute is set to TRUE.
- When you view a widget contained by one or more ancestor frames that are invisible, the VISIBLE attribute is set to TRUE and the HIDDEN attribute is set to FALSE for both the viewed widget and all its ancestor

frames. However, if the containing window or an ancestor window has its HIDDEN attribute set to TRUE, neither the viewed widget nor its ancestor frames are displayed.

- When you view a frame, that frame and all widgets contained within it are displayed except those widgets whose HIDDEN attributes are set to TRUE.
- When you view a window, the AVM displays that window and any ancestor windows only if no ancestor window has its HIDDEN attribute set to TRUE. If the AVM displays the window, it also views any descendant windows down to, but not including, the first descendant window that has its HIDDEN attribute set to TRUE.
- If you are displaying a root frame and there is not enough room in the window for the new root frame to display, the AVM removes other root frames, starting from the bottom of the window, until there is room for the new root frame.
- In the case of a PAGE-TOP or PAGE-BOTTOM frame, the VIEW statement activates the frame for display at the beginning or end of each page.

See also

DEFINE STREAM statement, HIDE statement, Widget phrase, Stream object handle

---

VIEW-AS phrase

Defines a compile-time defined (static) widget to represent a field or variable on the screen.

Note: Does not apply to SpeedScript programming.

Syntax

```
VIEW-AS
  {    combo-box-phrase
     | editor-phrase
     | FILL-IN
          [ NATIVE ]
          [   size-phrase ]
          [ TOOLTIP tooltip ]
     | radio-set-phrase
     | selection-list-phrase
     | slider-phrase
     | TEXT
          [ size-phrase ]
          [ TOOLTIP tooltip ]
     | TOGGLE-BOX
          [ size-phrase ]
          [ TOOLTIP tooltip ]
  }
```

Use the following syntax to specify the type of widget to use for displaying values in a browse column:

```
VIEW-AS combo-box-phrase | TOGGLE-BOX
```

combo-box-phrase

Specifies that a field or variable is viewed as a combo box widget. You can use a combo box to represent a value of any data type. This is the syntax for *combo-box-phrase*:

```
VIEW-AS COMBO-BOX
   [ LIST-ITEMS item-list | LIST-ITEM-PAIRS item-pair-list ]
   [ INNER-LINES lines ] [ size-phrase ] [ SORT ]
   [ TOOLTIP tooltip ]
   [ SIMPLE | DROP-DOWN | DROP-DOWN-LIST ]
   [ MAX-CHARS characters ]
   [ AUTO-COMPLETION [ UNIQUE-MATCH ] ]
```

Use the following syntax to specify a combo-box widget for displaying values in a browse column:

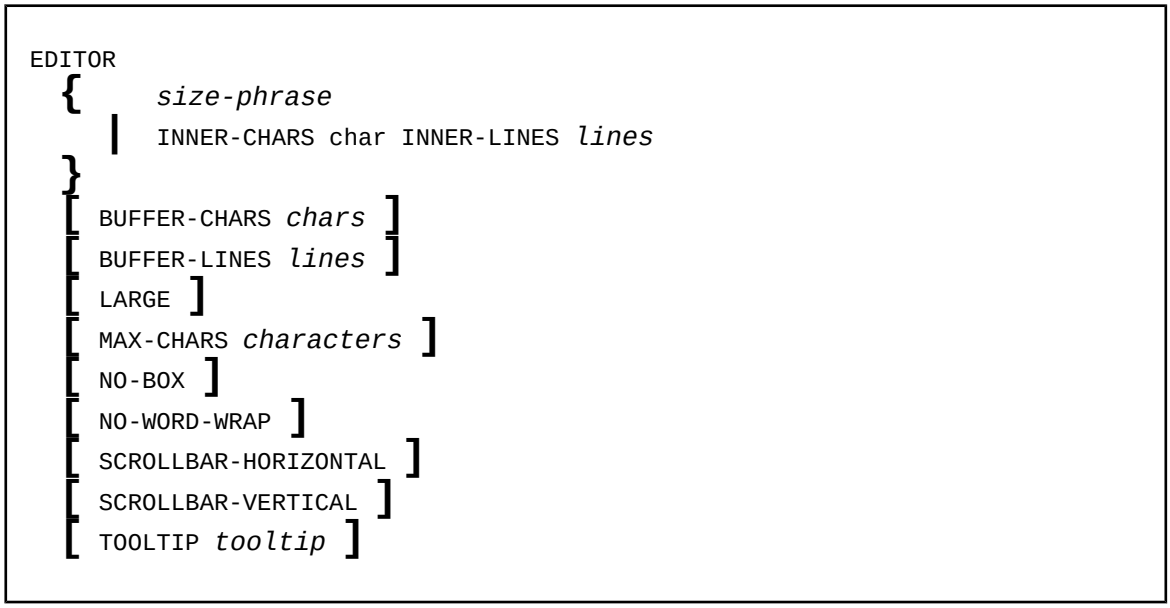
```
COMBO-BOX
   [ LIST-ITEMS item-list | LIST-ITEM-PAIRS item-pair-list ]
   [ INNER-LINES lines ] [ SORT ]
   [ DROP-DOWN | DROP-DOWN-LIST ]
   [ MAX-CHARS characters ]
   [ AUTO-COMPLETION [ UNIQUE-MATCH ] ]
```

Note: In a character interface, browse columns are always displayed as fill-in widgets no matter what VIEW-AS option you specify.

For more information, see the COMBO-BOX phrase reference entry.

editor-phrase

Specifies that a CHARACTER or LONGCHAR field or variable is viewed as a text editor widget. A text editor widget supports cut, paste, word-wrap, and auto-indent features. This is the syntax for *editor-phrase*:



For more information, see the EDITOR phrase reference entry.

`FILL-IN [ NATIVE ] [ size-phrase ]`

Specifies that the field or variable is viewed as a fill-in widget. In a fill-in field, the literal value of the field or variable is displayed. On update, the user types the literal value into the fill-in field.

You can specify FILL-IN for any CHARACTER, INT64, INTEGER, DECIMAL, DATE, DATETIME, DATETIME-TZ, or LOGICAL value (with or without extents). FILL-IN is the default representation for those values.

Note that Windows allows a user to transfer focus to the fill-in field by pressing **ALT** and one of the letters in the label. For more information on specifying a label using the LABEL option, see the Format phrase reference entry.

If you specify NATIVE, then the field behaves like a native fill-in field under the current user interface. A non-NATIVE field behaves like a default ABL fill-in field under any interface. Native fill-in fields provide better consistency with other applications in graphical environments, but do not support some ABL constructs such as the UPDATE statement with the TEXT option or the CHOOSE statement.

When a non-NATIVE (ABL) fill-in is disabled, the border disappears, but the text does not gray out. When a NATIVE fill-in is disabled, the text grays out.

Like the other static widgets that can be defined using the VIEW-AS phrase, you can specify ToolTips for the fill-in widget using the TOOLTIP option.

radio-set-phrase

Specifies that the field or variable is viewed as a radio set widget. A radio button set is a series of buttons, of which only one can be TRUE at a time. When the user sets one of the buttons to TRUE, the others are set to FALSE. You can specify a *radio-set-phrase* for any group of CHARACTER, INT64, INTEGER, DECIMAL, DATE, or LOGICAL values (with or without extents). This is the syntax for *radio-set-phrase*:

```
RADIO-SET
  [ HORIZONTAL [ EXPAND ] | VERTICAL ]
  [ size-phrase ]
  RADIO-BUTTONS label , value [ , label, value ... ]
  [ TOOLTIP tooltip ]
```

Note: If two or more buttons of a radio set use the same label, the AVM uses only the value of the first button.

For more information, see the RADIO-SET phrase reference entry.

selection-list-phrase

Specifies that the field or variable is viewed as a selection list widget. You can only specify the *selection-list-phrase* for a character-string value. A selection list is a scrollable list of CHARACTER values. If the field is enabled for input, the user can select one or more values from the list. This is the syntax for *selection-list-phrase*:

```
SELECTION-LIST
  [ SINGLE | MULTIPLE ]
  [ NO-DRAG ]
  [ LIST-ITEMS item-list ]
  [ SCROLLBAR-HORIZONTAL ]
  [ SCROLLBAR-VERTICAL ]
  { size-phrase | INNER-CHARS cols INNER-LINES rows }
  [ SORT ]
  [ TOOLTIP tooltip ]
```

For more information, see the SELECTION-LIST phrase reference entry.

slider-phrase

Specifies that the field or variable is viewed as a slider. Specify the *slider-phrase* for an integer value only. A slider is a graphical representation of a numeric range. It is composed of a rectangular area that contains a trackbar. You can change the current value within a defined range by moving the pointer that resides on the trackbar. This is the syntax for *slider-phrase*:

```
VIEW-AS SLIDER
  MAX-VALUE max-value MIN-VALUE min-value
  [ HORIZONTAL | VERTICAL ]
  [ NO-CURRENT-VALUE ]
  [ LARGE-TO-SMALL ]
  [ TIC-MARKS
      { NONE | TOP | BOTTOM | LEFT | RIGHT | BOTH }
      [ FREQUENCY n ]
  ]
  [ TOOLTIP tooltip ]
  [ size-phrase ]
```

For more information, see the SLIDER phrase reference entry.

TEXT [*size-phrase*]

Specifies that the field or variable is viewed as read-only text. In a graphical environment, a text field takes up less space on the screen than a native fill-in field.

You can specify TEXT for any CHARACTER, INT64, INTEGER, DECIMAL, DATE, or LOGICAL value (with or without extents).

TOGGLE-BOX [*size-phrase*]

Specifies that the field or variable is viewed as a toggle box widget. A toggle box is a small box that is either marked or not marked to indicate a TRUE or FALSE value, respectively. You can specify TOGGLE-BOX for any LOGICAL value, or any calculated value whose expression evaluates to a LOGICAL value.

Note that Windows allows a user to select a toggle-box item by pressing **ALT** and one of the letters in the side label. For more information on specifying a label using the LABEL option, see the Format phrase reference entry.

For toggle-box browse columns, the toggle-box always appears whether it is enabled or not. A solid outline appears to indicate that a cell in a toggle-box column currently has focus. The toggle-box is always centered within the column. You cannot specify *size-phrase* for toggle-box browse columns.

Note: In a character interface, browse columns are always displayed as fill-in widgets no matter what VIEW-AS option you specify.

TOOLTIP *tooltip*

Allows you to define a help text message for a toggle box. The AVM automatically displays this text when the user pauses the mouse over the toggle-box.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the ToolTip is removed. No ToolTip is the default. The TOOLTIP option is supported in Windows only.

Example

The following procedure defines a character variable and views it in succession as a text widget, a fill-in widget, an editor widget, and finally as a text widget again. The procedure shows that you can represent a character variable in several ways, as long as each representation appears in a separate frame.

r-viewas.p

```
DEFINE VARIABLE test AS CHARACTER NO-UNDO INITIAL "Now is the time"
  FORMAT "x(30)".

DISPLAY test VIEW-AS TEXT LABEL "Labels cannot be changed"
  WITH FRAME a SIDE-LABELS.
PAUSE.

UPDATE test VIEW-AS FILL-IN LABEL "But fillins can, please enter a new value"
  WITH FRAME b SIDE-LABELS.

UPDATE test VIEW-AS EDITOR INNER-CHARS 16 INNER-LINES 2 MAX-CHARS 70
  LABEL  "As can editors, please enter a new value:"
  WITH FRAME c.

DISPLAY test VIEW-AS TEXT FORMAT "x(70)" LABEL "The final value is:"
  WITH FRAME d.
```

For additional examples, see the COMBO-BOX phrase, EDITOR phrase, RADIO-SET phrase, SELECTION-LIST phrase, and SLIDER phrase reference entries.

Notes

- To create a static widget, you must define a static frame that contains the widget. Each frame you define that contains the widget creates an additional instance of that widget for the underlying field or variable. The handle for a static widget is not available until the widget is created.
- You can also use the VIEW-AS option in the Frame phrase and MESSAGE statement to indicate a dialog box and alert box, respectively.
- In Windows, if no font is specified for a fill-in field, ABL uses two default fonts:
  - A fixed font for date fields, numeric fields, and character fields that contain fill characters (such as the parentheses surrounding the area code of a telephone number).
  - A proportional font for character fields that do not contain fill characters.

  The AVM looks for these fonts in the current environment, which may be the registry (Windows only) or an initialization file. If the current environment does not define these fonts, the AVM uses the system default fixed and proportional fonts. For more information on environments, see *Manage ABL Applications*.

See also

COMBO-BOX phrase, EDITOR phrase, RADIO-SET phrase, SELECTION-LIST phrase, SIZE phrase, SLIDER phrase

WAIT-FOR statement (ABL only)

This WAIT-FOR statement instructs the AVM to stop executing the current block until a specific ABL event occurs. The AVM continues to respond to all other incoming events and execute any associated triggers or event procedures while in this wait state.

Note: ABL supports a separate version of the WAIT-FOR statement (.*NET WAIT-FOR*) that blocks for .NET, as well as ABL, events. For more information, see the WAIT-FOR statement (.NET and ABL) reference entry.

Syntax

```
WAIT-FOR event-list OF widget-list
   [ OR event-list OF widget-list ] ...
   [ FOCUS widget ]
   [ PAUSE n ]
```

event-list

A space or comma-separated list of user-interface events and other ABL events to wait for.

An event can be any event described in the Handle-based Object Events Reference.

widget-list

A space- or comma-separated list of widgets with which the event is associated. For more information on referencing widgets, see the Widget phrase reference entry.

FOCUS *widget*

Specifies the widget that initially receives input focus when the WAIT-FOR statement is executed. The value *widget* must be a valid reference to a widget (a widget name or handle) that is currently displayed and enabled.

PAUSE *n*

Specifies a time-out interval for the WAIT-FOR statement. The value *n* can be any numeric expression, including a fractional value. You can choose a whole or a fractional value for the time-out interval. If the time-out period you specify is a fractional value, the value is rounded to the nearest whole millisecond. If a period of *n* seconds elapses between events, the WAIT-FOR automatically terminates.

Examples

This procedure defines two buttons, defines triggers for them, and enables them. The procedure then waits for the user to close the current window. The initial focus is placed on the button labeled MORE. The user can then choose buttons continuously until closing the window or exiting with the **END-ERROR** key.

r-wait.p

```
DEFINE BUTTON more-button LABEL "MORE".
DEFINE BUTTON next-button LABEL "NEXT".

FORM Customer.CustNum Customer.Name more-button next-button
  WITH FRAME brief.

FORM Customer EXCEPT Customer.CustNum Customer.Name
  WITH FRAME full.

ON CHOOSE OF more-button
  DISPLAY Customer EXCEPT Customer.CustNum Customer.Name WITH FRAME full.

ON CHOOSE OF next-button DO:
  HIDE FRAME full.
  FIND NEXT Customer NO-LOCK NO-ERROR.
  IF AVAILABLE Customer THEN
    DISPLAY Customer.CustNum Customer.Name WITH FRAME brief.
END.

FIND FIRST Customer NO-LOCK.
DISPLAY Customer.CustNum Customer.Name WITH FRAME brief.

ENABLE more-button next-button WITH FRAME brief.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW FOCUS more-button.
```

If the user closes the current window then execution continues after the WAIT-FOR statement. In this case, the procedure ends because there are no more statements.

The following procedure uses the PAUSE option of the WAIT-FOR statement so that you automatically jump ahead to the next record if the user does not perform any action within three seconds after the customer information is displayed:

r-waitpn.p

```
DEFINE VARIABLE jump-ahead AS LOGICAL NO-UNDO INITIAL TRUE.

DEFINE BUTTON more-button LABEL "MORE".
DEFINE BUTTON next-button LABEL "NEXT".

FORM Customer.CustNum Customer.Name more-button next-button
  WITH FRAME brief.
FORM Customer EXCEPT Customer.CustNum Customer.Name
  WITH FRAME full.

ON CHOOSE OF more-button DO:
  DISPLAY Customer EXCEPT Customer.CustNum Customer.Name WITH FRAME full.
  jump-ahead = FALSE.
END.

ON CHOOSE OF next-button DO:
  jump-ahead = TRUE.
END.

ON WINDOW-CLOSE OF CURRENT-WINDOW DO:
  QUIT.
END.

ENABLE more-button next-button WITH FRAME brief.

DO WHILE TRUE:
  IF jump-ahead THEN
    RUN next-cust.

  WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW OR CHOOSE OF next-button
    FOCUS more-button PAUSE 3.
END.

PROCEDURE next-cust:
  HIDE FRAME full.
  FIND NEXT Customer NO-ERROR.
  IF AVAILABLE Customer THEN
    DISPLAY Customer.CustNum Customer.Name WITH FRAME brief.
END.
```

In this example, the code for finding the next Customer has been moved to an internal procedure. The WAIT-FOR statement has been placed inside a DO loop. The loop iterates when the user chooses the NEXT button or three seconds elapse. (If the user closes the window, the QUIT statement is executed and the loop does not iterate.) On each iteration, if the variable jump-ahead is TRUE, then the next-cust procedure is run to find and display the next Customer. If the user chooses the MORE button for a Customer, jump-ahead is set to FALSE. This prevents the procedure from automatically jumping ahead to the next Customer. Instead, the user can spend time examining the data. To move ahead to the next Customer, the user must explicitly choose the NEXT button. At that point, jump-ahead is reset to TRUE.

Notes

- Any widget associated with an event must be enabled before you wait on it.
- In general, do not use an UPDATE statement in an application that executes a WAIT-FOR statement. One exception is updating fields in a dialog box.
- In general, if you nest two WAIT-FOR statements in a single ABL application (where the nested WAIT-FOR executes in a trigger), you must ensure that your application satisfies the nested WAIT-FOR first. The event that satisfies the outer WAIT-FOR statement should be the terminating event for your application.
- In general, when a modal dialog box is active, the *event-list* can reference only events supported by the active dialog box and the widgets it contains. There are two exceptions:

  - You can specify an event on a procedure handle as long as *widget-list* specifies only a single procedure handle.
  - You can specify the PROCEDURE-COMPLETE event on an asynchronous request handle.

- In batch mode (running with the -b startup parameter) and when running on an application server, the WAIT-FOR statement supports only the following events:
  - PROCEDURE-COMPLETE
  - READ-RESPONSE
  - CONNECT
  - Developer events (U1 to U10, and CLOSE)

  The ten miscellaneous developer events (U1 to U10) have no standard meaning or action. They are reserved for an ABL developer to use for associating a trigger with some event that only happens programmatically using the APPLY statement. They have nothing to do with a particular user action and have no built-in significance. The CLOSE event has these same characteristics and only occurs when specified with the APPLY statement.

  OpenEdge clients running in batch mode process these events just like non-batch clients do.

- The PROCEDURE-COMPLETE event occurs for an asynchronous request handle when the current ABL session receives the response message from the application server that executed the request. When the WAIT-FOR statement executes, it processes any PROCEDURE-COMPLETE event that has occurred but has not yet been processed.
- PROCEDURE-COMPLETE events from a single application server connection are processed in the order that the associated asynchronous requests were originally generated. To ensure that all pending PROCEDURE-COMPLETE events are handled by a single WAIT-FOR statement, specify a single PROCEDURE-COMPLETE event for the last asynchronous request handle generated before the WAIT-FOR statement.
- To process a PROCEDURE-COMPLETE event for a particular asynchronous request handle, the AVM:
  - Decrements the ASYNC-REQUEST-COUNT attribute for the server referenced by SERVER attribute for the asynchronous request handle.
  - Decrements the ASYNC-REQUEST-COUNT attribute for a persistent procedure, if the PERSISTENT-PROCEDURE attribute of the asynchronous request handle refers to a valid persistent procedure.
  - Sets the COMPLETE attribute for the asynchronous request handle to TRUE.
  - Sets the STOP, QUIT, and ERROR attributes for the asynchronous request handle appropriately as indicated by the response message from the application server.
  - Sets the return value for the RETURN-VALUE function, if a return value was returned by the application server.
  - Stores any error information returned from the application server in the ERROR-STATUS system handle.
  - Attempts to execute the event procedure or method specified by the EVENT-HANDLER and the EVENT-HANDLER-CONTEXT attributes for the asynchronous request handle, if EVENT-HANDLER is not the empty string ("").
  - Sets each INPUT parameter for the event procedure or method to the Unknown value (?) or, if the parameter is a TEMP-TABLE, the TEMP-TABLE remains unchanged, if the response message indicates that the remote request finished with a STOP, ERROR, or QUIT condition.

  - Sets the INPUT parameter values for the event procedure or method to the OUTPUT and INPUT-OUTPUT parameter values returned by the remote procedure, if the response message indicates that the remote request completed successfully.
  - Displays an error message, if the specified event procedure or method fails to execute for any reason.
  - Raises any unhandled STOP condition, ERROR condition, or QUIT condition in the context of the WAIT-FOR statement, if the event procedure or method completes execution with that condition.

- These are possible causes for failing to execute the event procedure or method for a PROCEDURE-COMPLETE event. All of these failures raise a STOP condition in the context of the WAIT-FOR statement:
  - The procedure handle or method referenced by the EVENT-HANDLER-CONTEXT attribute is no longer valid.
  - The procedure or method specified by the EVENT-HANDLER attribute cannot be found.
  - The parameters to the procedure or method specified by the EVENT-HANDLER attribute are not all INPUT parameters.
  - The parameter signature of the procedure or method specified by the EVENT-HANDLER attribute does not match the output parameters returned in the response message for the asynchronous request.

- For SpeedScript, the WAIT-FOR statement instructs WebSpeed to stop executing the current block until the WEB-NOTIFY event occurs. The WEB-NOTIFY event is intended for internal use only, it does not apply to SpeedScript programming.
- In the context of the .NET blocking method, `System.Windows.Forms.Application:Run( )`, if you directly or indirectly execute this WAIT-FOR statement while displaying a non-modal ABL window, in certain contexts (such as within a user-defined function or non-`VOID` method) this statement execution raises the STOP condition. For more information on the .NET `Application:Run( )`, method, see the reference entry for the WAIT-FOR statement (.NET and ABL).

See also

CONNECT event, Developer events, DISABLE statement, ENABLE statement, ON statement, PROCEDURE-COMPLETE events, READ-RESPONSE event, Trigger phrase, WAIT-FOR statement (.NET and ABL), Widget phrase

---

WAIT-FOR statement (.NET and ABL)
(Windows only; GUI for .NET only)

This WAIT-FOR statement instructs the AVM to stop executing the current block and remain in a wait state (blocking) until a .NET method that it calls returns. The AVM continues to respond to all incoming ABL events (see the WAIT-FOR statement (ABL only) reference entry) as well as .NET events, and it executes any associated triggers, event procedures, or .NET event handlers while in this wait state.

Note: ABL supports a separate version of the WAIT-FOR statement (*ABL WAIT-FOR*) that blocks only for ABL events. For more information, see the WAIT-FOR statement (ABL only) reference entry.

Syntax

```
WAIT-FOR { object-reference | type-name } : method-name ( [ parameters ] )
  [ SET return-value ]
```

object-reference

A reference to an object that generally inherits from the .NET class `System.Windows.Forms.Form` or `System.Windows.Forms.CommonDialog`. OpenEdge provides a particular subclass of `System.Windows.Forms.Form`—the Progress.Windows.Form class—which you can use to create .NET forms in an ABL session that co-exist more naturally with ABL windows.

type-name

The type name of a .NET class that provides a static blocking method, most commonly `System.Windows.Forms.Application`. With an appropriate USING statement, you can also specify the type by its unqualified class name (Application).

method-name

The name of a .NET input-blocking method that the WAIT-FOR statement calls, which is typically `Run` (a static method on `System.Windows.Forms.Application`), or by .NET convention, `ShowDialog` (an instance method on `System.Windows.Forms.Form` or `System.Windows.Forms.CommonDialog`).

parameters

Parameters for the method specified by *method-name*.

SET *return-value*

Provides the return value from the method, *method-name*( ), which is set when the WAIT-FOR statement completes execution. The *return-value* can be a variable, property, or field that has the same data type as the *method-name*( ) return value, typically `System.Windows.Forms.DialogResult`.

To use this option, *method-name*( ) must be a non-VOID method. If you specify this option for a VOID method, such as `System.Windows.Forms.Application:Run( )`, ABL raises a compile-time error.

For more information on .NET input-blocking methods that you can call in the WAIT-FOR statement, see the notes of this reference entry.

Example

The ABL-derived .NET class, `r-WaitForms`, inherits the Progress.Windows.Form class to implement a non-modal .NET form. When you try to close the displayed form, a dialog box appears that prompts if you want the form to complete closing or not. If you choose to complete closing, the form closes. If you choose to cancel the closing, the form remains displayed, and you can try to close the form, again.

When you instantiate `r-WaitForms`, it initializes and subscribes a handler (the `Form_Closing( )` method) to the `FormClosing` event of the form. You can then display the form by calling the `DoWait( )` method on the `r-WaitForms` instance. This method executes the WAIT-FOR statement, which calls the .NET input-blocking method `System.Windows.Forms.Application:Run( )`. (For more information on this method, see the notes.) When you try to close the displayed form, this causes the non-modal form to publish its `FormClosing` event, which executes the `Form_Closing( )` method to handle the event.

r-WaitForms.cls

```
USING System.Windows.Forms.* FROM ASSEMBLY.
USING Progress.Util.*        FROM ASSEMBLY.

CLASS r-WaitForms INHERITS Progress.Windows.Form:

  DEFINE VARIABLE rFormDescr AS CLASS Label NO-UNDO.

  METHOD PUBLIC VOID DoWait( ).
    /* Display and wait for the non-modal form to close */
    WAIT-FOR Application:Run( INPUT THIS-OBJECT ).
  END METHOD.

  CONSTRUCTOR PUBLIC r-WaitForms( ):
    /* Initialize and subscribe to events */
    InitializeComponent( ).
    THIS-OBJECT:FormClosing:Subscribe(Form_Closing).
  END CONSTRUCTOR.

  METHOD PRIVATE VOID InitializeComponent( ):
    /* Initialize the non-modal form class and components */
    rFormDescr = NEW Label( ).

    /* Initialize the form description label */
    rFormDescr:Text     = "Click the Close (X) button of this form to
                          pop-up
                          a dialog box ...".
    rFormDescr:Size     = NEW System.Drawing.Size( INPUT 330, INPUT 13 ).
    rFormDescr:Location = NEW System.Drawing.Point( INPUT 4, INPUT 6 ).

    /* Initialize the non-modal form */
    THIS-OBJECT:FormBorderStyle = FormBorderStyle:FixedSingle.
    THIS-OBJECT:Text = "This is my form.".
    THIS-OBJECT:Controls:Add( INPUT rFormDescr ).
    THIS-OBJECT:Size = NEW System.Drawing.Size( INPUT rFormDescr:Width, INPUT 60 ).
  END METHOD.

  METHOD PRIVATE VOID Form_Closing
    ( INPUT sender AS System.Object, INPUT e AS FormClosingEventArgs ):
    DEFINE VARIABLE rDialog       AS CLASS Progress.Windows.Form NO-UNDO.
    DEFINE VARIABLE rDialogDescr  AS CLASS Label NO-UNDO.
    DEFINE VARIABLE rOKButton     AS CLASS Button NO-UNDO.
    DEFINE VARIABLE rCancelButton AS CLASS Button NO-UNDO.
    DEFINE VARIABLE enDialogResult AS CLASS DialogResult NO-UNDO.

    /* Create dialog box components */
    ASSIGN
      rDialog       = NEW Progress.Windows.Form( )
      rDialogDescr  = NEW Label( )
      rOKButton     = NEW Button( )
      rCancelButton = NEW Button( ).

    /* Initialize the dialog description label */
    rDialogDescr:Text     = "Click OK to close form or click Cancel to
                            leave form open.".
    rDialogDescr:Size     = NEW System.Drawing.Size( INPUT 306, INPUT 13).
    rDialogDescr:Location = NEW System.Drawing.Point( INPUT 4, INPUT 6 ).

    /* Initialize the buttons */
    rOKButton:Text = "OK".
    rOKButton:Size = NEW System.Drawing.Size( INPUT 60, INPUT 20).
    rOKButton:Location = NEW System.Drawing.Point
      ( INPUT INTEGER( ( rDialogDescr:Width - 124 ) / 2 ),
        INPUT rDialogDescr:Top + rDialogDescr:Height + 8 ).
    rOKButton:DialogResult = DialogResult:OK.
    rCancelButton:Text = "Cancel".
    rCancelButton:Size = NEW System.Drawing.Size( INPUT 60, INPUT 20 ).
```

```
    rCancelButton:Location = NEW System.Drawing.Point
      ( INPUT rOKButton:Left + rOKButton:Width + 4,
        INPUT rDialogDescr:Top + rDialogDescr:Height + 8 ).
    rCancelButton:DialogResult = DialogResult:Cancel.

    /* Initialize the modal dialog box with label and buttons */
    rDialog:FormBorderStyle = FormBorderStyle:FixedDialog.
    rDialog:Controls:Add( INPUT rDialogDescr ).
    rDialog:Controls:Add( INPUT rOKButton ).
    rDialog:Controls:Add( INPUT rCancelButton ).
    rDialog:Text = "My form is closing ...".
    rDialog:Size = NEW System.Drawing.Size( INPUT 306, INPUT 106 ).

    /* Display dialog box to handle FormClosing event and the results */
    WAIT-FOR rDialog:ShowDialog( ) SET enDialogResult.

    IF EnumHelper:AreEqual
      ( INPUT enDialogResult, INPUT DialogResult:Cancel ) THEN DO:
      MessageBox:Show( INPUT "My form closing was canceled." ).
      e:Cancel = TRUE. /* Cancel FormClosing; leave the main form open */
    END.
    ELSE DO:
      MessageBox:Show( INPUT "My form is closing OK." ).
      e:Cancel = FALSE. /* Continue FormClosing; close the main form */
    END.

  END METHOD. /* Form_Closing */

END CLASS.
```

The `Form_Closing( )` method passes INPUT parameters from .NET for the `FormClosing` event. One of these parameters (e) is a `System.Windows.Forms.FormClosingEventArgs` object, which contains a Cancel property whose setting allows the event handler to either complete the `FormClosing` event or interrupt and cancel the `FormClosing` event. To determine how to set this property, the event handler instantiates, initializes, and displays another Progress.Windows.Form class (`rDialog`) as a modal dialog box.

The dialog box contains two buttons, `rOKButton` and `rCancelButton`, whose `DialogResult` properties are set to the `System.Windows.Forms.DialogResult` enumeration values OK and Cancel, respectively. The event handler displays `rDialog` as a modal form by executing the WAIT-FOR statement, which calls the modal input-blocking method `System.Windows.Forms.Form:ShowDialog( )`. (For more information on this method, see the notes.)

When you click one of the two dialog buttons, this causes the dialog box to close and the `ShowDialog( )` method to return. This automatically sets the `DialogResult` property on `rDialog` to the value of the `DialogResult` property on the button that you have clicked and also returns the same property value as the value of `ShowDialog( )`, which the WAIT-FOR statement assigns to the variable, `enDialogResult`. The event handler then uses the static `AreEqual( )` method on the Progress.Util.EnumHelper class to test the value of `enDialogResult` and set the `e:Cancel` property to either complete the `FormClosing` event or cancel the `FormClosing` event and leave the non-modal form open for further input.

Note: The calls to `System.Windows.Forms.MessageBox:Show( )` display a message box similar to the ABL MESSAGE statement with the VIEW-AS ALERT-BOX option.

To instantiate `r-WaitForms` and display the non-modal form, you can thus run a procedure with code like this:

```
DEFINE VARIABLE rWaitForms AS CLASS r-WaitForms NO-UNDO.

rWaitForms = NEW r-WaitForms( ).
rWaitForms:DoWait( ).
```

Notes

- A non-GUI ABL session cannot visualize .NET form and control objects and cannot block for GUI or non-GUI .NET events. So, you cannot invoke this WAIT-FOR statement (calling a .NET input-blocking method) within a non-GUI, Windows ABL session, including a:
  - Character mode (CHUI) client
  - Batch-mode client
  - Application server session
- If you use any .NET forms in an ABL session, you can execute only **one** .NET WAIT-FOR statement that processes events for all .NET non-modal forms and their controls. This statement must be the first WAIT-FOR statement for processing non-modal events in your application. Following this statement, from event handlers and trigger blocks, you can execute multiple input-blocking statements to process any modal .NET form or ABL dialog box. ABL events for non-modal windows and their child widgets, or for non-GUI ABL features, such as asynchronous remote procedure calls and socket operations, all work in the context of this single non-modal .NET WAIT-FOR statement.

  Caution: Using stacked input-blocking statements other than for dialog boxes is not recommended. If you do so, you must take special care to ensure that any WAIT-FOR statements that you simultaneously execute to process non-modal events complete their execution in reverse order of invocation. Otherwise, your application will have unpredictable behavior. Also, in certain contexts (such as within a user-defined function or non-VOID method), if you execute an additional input-blocking or event-processing statement while displaying an ABL non-modal window **together with** .NET non-modal forms, the AVM raises the STOP condition.

- To block on any number of non-modal .NET forms (and ABL windows), you must use a WAIT-FOR statement that executes the `System.Windows.Forms.Application:Run( )` input-blocking method (as shown for the example class, `r-WaitForms`):

  ```
  WAIT-FOR System.Windows.Forms.Application:Run ( [ form-object-ref ] )
  ```

  You can specify *`form-object-ref`* as an object reference to a single .NET non-modal form object, on which the WAIT-FOR statement blocks, displays, and waits to close. If you specify *`form-object-ref`*, the statement also displays any additional non-modal forms that you have previously initialized by setting their Visible properties to TRUE or by invoking their `Show( )` methods. However, .NET automatically displays *`form-object-ref`*, itself, without having to set its Visible property or run its `Show( )` method. You can also use triggers, event procedures, and .NET event handlers to create and display additional non-modal .NET forms (or ABL windows) after the WAIT-FOR statement blocks for events.

The conditions for unblocking this WAIT-FOR statement differ, depending on whether you specify *form-object-ref*. However, whatever conditions unblock the WAIT-FOR statement also automatically close any open .NET forms. This is different from ABL windows, which you must explicitly open and close using ABL statements or widget attributes without regard to the execution of a WAIT-FOR statement.

With *form-object-ref*, the WAIT-FOR statement unblocks and continues execution with the following statement if one of the following actions occurs:

- Your application explicitly calls the *form-object-ref*`:Close( )` method, which automatically calls the `Dispose( )` method to garbage collect the form.
- The user clicks the form Close (**X**) button in the upper right corner of the *form-object-ref* form, and you do not cancel the action in a handler for the `FormClosing` event, as in the example. This action also automatically calls the *form-object-ref*`:Close( )` method.
- Your application calls the `System.Windows.Forms.Application:Exit( )` method.

When you use any of these techniques to unblock the WAIT-FOR statement, all currently displayed non-modal forms close, in addition to the form specified by *form-object-ref*, and including any non-modal .NET forms that were created and displayed after the WAIT-FOR statement blocked.

Caution:

If you are executing the READKEY statement within a trigger or event handler while blocking on a *form-object-ref*, and the user clicks the form Close (**X**) button, the ABL application shuts down unconditionally. For example, the following READKEY loop can cause this shutdown to occur:

```
DO WHILE LASTKEY != KEYCODE("F3"):
   READKEY.
   IF LASTKEY = KEYCODE("F3") THEN RETURN.
END.
```

This shutdown occurs because .NET generates a WM_QUIT message in response to clicking the Close (**X**) button that READKEY interprets (by design) as a message to shut down the application.

Caution: You might have a problem displaying message boxes after invoking `Application:Exit( )` or `Application:ExitThread( )`. One way this can happen: if you use any two of the techniques for unblocking the WAIT-FOR, such as by calling `Close( )`, then calling `Application:Exit( )`, any subsequent executions of the MESSAGE statement with the VIEW-AS ALERT-BOX option or any subsequent calls to `System.Windows.Forms.MessageBox:Show( )` might not display the specified message box and the user will hear a beep sound instead.

If you do not specify *form-object-ref*, the statement displays and blocks for input on any non-modal forms that you have previously initialized by setting their `Visible` properties to TRUE or by invoking their `Show( )` methods. Without *form-object-ref*, you also do not need to have a .NET form instantiated before you execute the WAIT-FOR statement. Without any non-modal .NET form created, this statement processes ABL events until you create and initialize your first .NET non-modal form for display in an associated event handler or trigger, at which point the same WAIT-FOR statement processes both .NET and ABL events.

Also, if you do not specify *form-object-ref*, the WAIT-FOR statement unblocks and continues execution with the following statement only when you invoke the `System.Windows.Forms.Application:Exit( )` method at some point in the ABL session. This method closes all non-modal .NET forms that are currently open before unblocking the WAIT-FOR statement. Note that using this technique, you must be sure to create and initialize at least one .NET form or ABL window (non-modal or modal) so there are active components to work with during the input-blocking state. Otherwise, the blocking WAIT-FOR statement blocks indefinitely or until the user presses **CTRL+BREAK**.

Note: When working with any displayed non-modal form, except the form specified by *form-object-ref*, the user (using the Close (**X**) button) or the application (using the Close( ) method) can close the form. However, this does not by itself unblock the blocking WAIT-FOR statement. If you want the WAIT-FOR to unblock other than by closing the form specified by *form-object-ref*, your application must call Application:Exit( ).

- To block on a .NET modal dialog box, you must use a WAIT-FOR statement that executes the ShowDialog( ) input-blocking method on the specified form object (as shown for rDialog in the example class, r-WaitForms.cls):

```
WAIT-FOR dialog-object-ref:ShowDialog ( [ parent-form ] )
```

This causes the WAIT-FOR statement to display the form specified by the *dialog-object-ref* object reference, and block for input on that form as a dialog box. You can also specify the object reference of a form (*parent-form*) that becomes the parent of the dialog box referenced by *dialog-object-ref*. (This allows the .NET dialog box to display centered over the parent form.)

Note that by convention, every .NET class that can implement a modal dialog box has a ShowDialog( ) method, such as System.Windows.Forms.Form and System.Windows.Forms.CommonDialog, as well as classes that derive from them, like System.Windows.Forms.OpenFileDialog. Therefore, this syntax for the WAIT-FOR statement allows you to open all these different types of dialog boxes.

Note: The System.Windows.Forms.MessageBox class does not work this way. Instead, it is similar to the MESSAGE statement with the VIEW-AS ALERT-BOX option, which handles its own input without events. Similarly, you can **only** open a MessageBox by invoking its static Show( ) method outside of a WAIT-FOR statement.

With *dialog-object-ref*:ShowDialog( ), the WAIT-FOR statement unblocks and continues execution with the following statement if one of the following actions occurs:

- The user clicks the dialog box Close (**X**) button in the upper right corner of the dialog box.
- You set the *dialog-object-ref*:DialogResult property (if the form class supports it) to a valid System.Windows.Forms.DialogResult enumeration value.
- You call the *dialog-object-ref*:Close( ) method. Note that calling this method also automatically calls the Dispose( ) method on the form, making the form object available for garbage collection.

For any of these actions, the FormClosing and FormClosed events are also published on *dialog-object-ref*, and you can handle the FormClosing event in order to prevent the form from being closed by canceling the action, as shown in the example.

Caution: .NET automatically calls Dispose( ) on a non-modal form when it is closed, unless the form is shown as a dialog box. .NET simply hides the dialog when it is closed. However, the AVM calls Dispose( ) by default for dialogs that inherit from Progress.Windows.Form. The AVM does not do this if the DisposeDialogOnClose property of the form is set to FALSE. This can be set on a particular instance or globally using the No Dialog Dispose (-nodialogdispose) startup parameter. One use case for not calling Dispose( ) is if you want to re-use the form at a later time.

At this point, you can check the user response to the dialog box. Note that `ShowDialog( )` returns a `DialogResult` (enumeration) value with the result of the dialog box. You can access this value using the SET option (as shown in the `r-WaitForms.cls` example) or by checking the *`dialog-object-ref`*`:DialogResult` property (if the form object is still available and the class supports it).

Note: Not all .NET form classes provide a public `DialogResult` property—for example, `System.Windows.Forms.FileDialog`. For a form that does not provide this property, you can handle dialog box results in the following ways: 1) by using the SET option to return the form's `ShowDialog( )` method value, 2) by using event handlers subscribed to the events that the form provides, for example, the `FileOk` or `HelpRequest` event, or 3) by testing the values of properties that might otherwise be set depending on input to the dialog box, such as the `FileName` property of the `System.Windows.Forms.OpenFileDialog` object.

.NET does not set the *`dialog-object-ref`*`:DialogResult` property automatically except in two cases:

- The user clicks the dialog box Close (**X**) button in the upper right corner, which sets the property to the `DialogResult:Cancel` enumeration value.
- The user clicks a button (or any control that implements the `System.Windows.Forms.IButtonControl` interface) contained by the dialog box whose `DialogResult` property you have set with a valid `DialogResult` enumeration value. In this case, .NET automatically sets the *`dialog-object-ref`*`:DialogResult` property to the value of the button property.

Otherwise, your application must set the value of *`dialog-object-ref`*`:DialogResult` directly, typically in an event handler. Note that if you want .NET to automatically set the *`dialog-object-ref`*`:DialogResult` property from a button `DialogResult` property, your application must initialize the `DialogResult` value for the button property before the user clicks a given button.

- You can detect that a specific form is closing by handling its `FormClosing` event; you can detect that a specific form has already closed by handling its `FormClosed` event. However, note that the `Closed` event does not fire for non-modal .NET forms that you close by calling `Application:Exit( )` and that are not specified by the *`form-object-ref`* parameter passed to the `Application:Run( )` method.

Caution: Do not delete the ABL object reference to the object (sender) that publishes a `FormClosing` event from within its `FormClosing` event handler. This causes the `FormClosing` event to be published a second time. If you need to delete the sender for an event that is associated with closing a form before the AVM garbage collects it, execute the DELETE OBJECT statement for the sender within a handler for its `FormClosed` event.

- If you specify a non-blocking method in a WAIT-FOR statement, the method executes and the WAIT-FOR immediately returns, resuming execution with the next available statement. ABL does not prevent this. However, the WAIT-FOR statement, in this case, works no differently than invoking the method as a statement by itself.

Note: Progress Software Corporation recommends that you **donot** invoke non-blocking methods using the WAIT-FOR statement.

- The one WAIT-FOR statement that you execute for non-modal .NET forms after setting their `Visible` properties to TRUE or by invoking their `Show( )` methods, must call the `Application:Run( )` method to make the forms visible and usable. If you execute any other form of the WAIT-FOR statement after making non-modal .NET forms visible, such as one that blocks for an ABL event (even a developer event, such as U1 OF THIS-PROCEDURE), the ABL virtual machine (AVM) raises STOP on this WAIT-FOR statement.

- You cannot set the `Visible` property or call the `Show( )` method on a modal .NET form before executing a WAIT-FOR statement on the `ShowDialog( )` method. If you execute a WAIT-FOR statement that calls the `ShowDialog( )` method on a modal .NET form that you have previously made visible, .NET raises a run-time exception.
- Once you execute a non-modal WAIT-FOR statement that calls the `Application:Run( )` method, the statement goes into a wait state that allows any event handlers or triggers to run in response to all types of ABL-supported events, including .NET events, ABL UI events, and ABL non-UI events (such as socket events). In the associated event handlers or triggers, you can then create and display additional .NET non-modal forms (by setting their `Visible` properties or invoking their `Show( )` methods) or ABL non-modal windows (for example, by setting their VISIBLE attributes to TRUE, executing DISPLAY statements, or executing a VIEW statement), and the existing WAIT-FOR statement processes events associated with these new non-modal .NET forms or ABL windows in addition to any other events it is already processing. To open modal dialog boxes (.NET or ABL) from an event handler or trigger, you must execute an additional WAIT-FOR statement for each dialog box that you open. Each such WAIT-FOR statement then blocks until its associated modal dialog box is closed, allowing the event handler or trigger that invoked the statement to resume execution.
- .NET does not support blocking for multiple non-modal forms simultaneously at more than one point in an application. If you attempt to execute more than one non-modal WAIT-FOR statement that calls `Application:Run( )` by invoking a second such WAIT-FOR statement in an event handler or trigger, .NET raises an exception, which in turn raises STOP on this second WAIT-FOR statement.

Caution: After you execute an additional WAIT-FOR statement that calls `Application:Run( )`, even if you trap the STOP condition with ON STOP, .NET does not allow another call to `Applicaton:Run( )` in the same session. You must exit the ABL session and fix the application to avoid such simultaneous calls to `Applicaton:Run( )`.

- You must execute a separate WAIT-FOR statement for each modal .NET dialog box or ABL dialog box that you open. The modal nature of each dialog box guarantees that the corresponding WAIT-FOR statements unblock and resume execution with the statements that follow them in order of the most recently opened dialog box. However, while blocking on any .NET or ABL modal dialog box, you cannot then execute a non-modal WAIT-FOR statement calling `Application:Run( )`, even if no previous non-modal WAIT-FOR statement is still in a wait state. Attempting to call a non-modal WAIT-FOR statement while any modal .NET dialog box or ABL dialog box is open raises a run-time error.
- If you add non-modal .NET forms to an existing ABL application that already contains WAIT-FOR statements to process non-modal ABL GUI and non-GUI events, replace these WAIT-FOR statements (**except** those that block on ABL modal dialog boxes) with a single WAIT-FOR statement that calls `Application:Run( )`. In order to have an active .NET form in your application, you must execute your latest (and usually only) `WAIT-FOR` statement in the format that calls a .NET input-blocking method (for example, waiting on `Application:Run( )`) so that it is in effect to process .NET events. This allows the application to handle the same ABL events after adding .NET forms that it handled before you added the .NET forms.
- You cannot use the APPLY statement to force a WAIT-FOR statement that is blocking on .NET forms to unblock and continue execution with the statement following the WAIT-FOR statement. Instead, call the appropriate .NET method (for example, `Close( )` or `Application:Exit( )`) to unblock the WAIT-FOR statement and continue with the next statement.
- To handle a particular .NET event while the WAIT-FOR statement that processes the event is executing, you must subscribe a .NET event handler to the specified event using the Subscribe( ) event method **before** the event is raised. For more information on event methods and event handlers for .NET events, see the Class Events Reference.
- If an ABL handler for a .NET event raises an unhandled error condition or throws an error object out of the handler, the AVM does not throw a .NET `Exception` back to the .NET Common Language Runtime (CLR), but displays a message to the default output device and continues blocking for events.

- You cannot invoke the non-modal .NET `System.Windows.Forms.Application:Run( )` method or the .NET `ShowDialog( )` method used to display a modal dialog box in any ABL context other than in a WAIT-FOR statement. Any attempt to do so raises a run-time error.

See also

Type-name syntax, USING statement, WAIT-FOR statement (ABL only)

WEEKDAY function

Evaluates a date expression and returns the day of the week as an INTEGER value from 1 (Sunday) to 7 (Saturday) for that date.

Syntax

```
WEEKDAY ( date )
```

```
WEEKDAY ( datetime-expression )
```

date

A date expression for which you want the day of the week.

datetime-expression

An expression that evaluates to a DATETIME or DATETIME-TZ. The WEEKDAY function returns the weekday of the date part of the DATETIME or DATETIME-TZ value.

Example

This procedure tells you the day of the week that you were born and how many days old you are:

r-wkday.p

```
DEFINE VARIABLE birth-date AS DATE      NO-UNDO
  LABEL "Birth Date".
DEFINE VARIABLE daynum     AS INTEGER   NO-UNDO.
DEFINE VARIABLE daylist    AS CHARACTER NO-UNDO FORMAT "x(9)"
  INITIAL "Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday".
DEFINE VARIABLE dayname    AS CHARACTER NO-UNDO
  LABEL "Day You Were Born".
DEFINE VARIABLE daysold    AS INTEGER   NO-UNDO
  LABEL "Days Since You Were Born".

REPEAT:
  SET birth-date.
  ASSIGN
    daynum  = WEEKDAY(birth-date)
    dayname = ENTRY(daynum, daylist)
    daysold = TODAY - birth-date.

  DISPLAY dayname daysold.
END.
```

See also

ADD-INTERVAL function, DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, ETIME function, INTERVAL function, ISO-DATE function, MONTH function, MTIME function, NOW function, TIME function, TIME-SOURCE attribute, TIMEZONE function, TODAY function, YEAR function, YEAR-OFFSET attribute

WIDGET-HANDLE function

This function is supported only for backward compatibility. Use the HANDLE function instead.

Converts a string representation of a handle to a valid handle.

Syntax

```
WIDGET-HANDLE ( handle-string )
```

Caution: Use this function only to convert a handle previously stored as a string value back to a valid handle. If you convert an arbitrary string to handle using this function and then reference the new handle, a system error will occur. If you use the VALID-HANDLE function to validate a handle generated from an arbitrary string value, a system error will occur.

handle-string

A string representation of a handle. Since handles are integer values, the string must contain only numeric characters.

Widget phrase

References a widget in a statement. The Widget phrase is used in the APPLY, ON, and WAIT-FOR statements.

Note: Does not apply to SpeedScript programming.

Syntax

```
{   FRAME frame
  | [ FIELD ] field [ IN FRAME frame ]
  | column [ IN BROWSE browse ]
  | { MENU | SUB-MENU } menu
  | MENU-ITEM menu-item [ IN MENU menu ]
  | handle
  | system-handle
}
```

`FRAME` *frame*

Specifies a frame widget. The *frame* parameter must be the name of an existing frame.

`[ FIELD ]` *field* `[ IN FRAME` *frame* `]`

Specifies a field. The FIELD keyword is optional. The *field* parameter must be the name of an existing field-level widget: a fill-in, editor, text, slider, toggle box, radio set, selection list, combo box, button, image, rectangle, or browse. Use the IN FRAME option to qualify the widget, if necessary.

column `[ IN BROWSE` *browse* `]`

Specifies a column or cell in a browse widget. Use the IN BROWSE option to qualify the widget, if necessary. For more information on when you can reference browse columns and cells, see the DEFINE BROWSE statement reference entry.

`{ MENU | SUB-MENU }` *menu*

Specifies a menu or submenu. The *menu* parameter must be the name of an existing menu. The menu can be a pop-up menu, pull-down menu, or menu bar. Within the widget phrase, ABL does not distinguish between MENU and SUB-MENU.

`MENU-ITEM` *menu-item* `[ IN MENU` *menu* `]`

Specifies an menu item within a menu. The menu item parameter must be the name of an existing menu item. Use the IN MENU option to qualify the menu item, if necessary.

handle

Variable or field that specifies a valid widget, procedure, or system handle.

system-handle

Specifies a built-in system handle. The system handle parameter must be one of the built-in system handles listed in the following table.

Table 64: System handles

System handle	Description
ACTIVE-WINDOW	A handle to the ABL window that has most recently received input focus during the session.
CLIPBOARD	A handle to the system clipboard.
COLOR-TABLE	A handle to information on the current color table.
COMPILER	A handle to information on the most recently executed COMPILE statement.
CURRENT-WINDOW	A settable handle to the default window for the ABL session.[59, 60]
DEBUGGER	A handle to the Application Debugger.
DEFAULT-WINDOW	A handle to the static window created by the AVM for the session. Every session has one static window.[59]
ERROR-STATUS	A handle to information on the last statement executed with the NO-ERROR option.
FILE-INFO	A handle to information on an operating system file.
FOCUS	A handle to the field-level widget that currently has keyboard focus (that is, the current field).
FONT-TABLE	A handle to information on the current font table.
LAST-EVENT	A handle to the last event received by the program.
RCODE-INFO	A handle to information on an ABL r-code file.
SELF	A handle for the widget associated with the currently executing user-interface trigger.
SESSION	A handle to information on the current ABL session.
SOURCE-PROCEDURE	A handle to the procedure file that contains the original invocation (RUN statement or function invocation) of the current internal procedure or user-defined function.

[59] The initial setting of the CURRENT-WINDOW handle is the ABL static window. CURRENT-WINDOW can also be set to the handle of any dynamic window.

[60] If the THIS-PROCEDURE:CURRENT-WINDOW attribute is set to the handle of a valid window, this window becomes the default window for the executing procedure (overriding the setting of the CURRENT-WINDOW handle). The setting of THIS-PROCEDURE:CURRENT-WINDOW changes the default window only for the current external procedure block.

System handle	Description
TARGET-PROCEDURE	From within an internal procedure: A handle to the procedure file mentioned, explicitly or implicitly, by the original RUN statement that invoked (perhaps through a chain of super procedures) the current internal procedure.From within a user-defined function: A handle to the procedure file mentioned, explicitly or implicitly, by the original function invocation that invoked (perhaps through a chain of super versions of functions) the current user-defined function.
THIS-PROCEDURE	A handle to the executing external procedure in which the handle is referenced.

Notes

For information on how to access widget attributes and methods, see the Accessing handle attributes and methods.

Working directory

A working directory is the default operating system directory where an ABL session reads or writes most data files (such as text and XML files) that are specified with a relative pathname. An ABL session can have one active working directory at a time that is the *current working directory*. In OpenEdge, the current working directory can be one of the following:

- When you run an ABL session that is started directly from a shortcut in the OpenEdge program group in Windows, such as the OpenEdge Desktop, the current working directory is the working directory that you specified during OpenEdge installation (`C:\OpenEdge\WRK` by default).
- When you run an ABL session (OpenEdge AVM) from within Progress Developer Studio for OpenEdge, the current working directory is the working directory you specified for the project you are currently working in.
- When you start an ABL client directly from the command line, for example, by running the OpenEdge `pro` or `mpro` command in the OpenEdge Proenv command window, the current working directory is the active working directory at the point you run the command.

Notes

- The current working directory is distinct and different from `PROPATH`. `PROPATH` is an environment variable that specifies one or more directories that an ABL session searches in order to locate and execute ABL code in any procedure file, include file, class definition file, or r-code file that you specify with a relative pathname.
- The ABL session does **not** search the current working directory for data files that you access in order to load the contents of an editor widget. Instead, if you pass a relative pathname to the INSERT-FILE( ) method or the READ-FILE( ) method, the ABL session searches `PROPATH` to locate and read the file into the associated editor widget. This is an exception to how ABL generally handles access to data files specified with a relative pathname.

See also

PROPATH function, PROPATH statement

XOR operator (bitwise)

Performs a bitwise exclusive OR operation on two expressions of the same flag enum type and returns a new instance of the same type.

Syntax

```
flag-enum-expression XOR flag-enum-expression
```

flag-enum-expression

An expression whose value is a flag enum instance. Both expressions must be the same flag enum type, and neither expression can be the Unknown value (?).

Example

When you perform an XOR operation on two flag enum instances, the resulting instance has a flag set if the state of that flag was different for the original two expressions. For example, using this enum type:

```
ENUM Alignment FLAGS:
    DEFINE ENUM Left
                Right
                Top
                Bottom.
END ENUM.
```

The following code fragment shows examples of two different XOR operations.

```
DEFINE myAlignment AS Alignment.

myAlignment = Alignment:Left.
/* The initial value for myAlignment has the Left set. */

myAlignment = myAlignment XOR Alignment:Top.
/* After the first XOR operation, myAlignment has both the Left and Top
   flags set because initially, myAlignment has the Left flag set and
   Alignment:Top does not, and vice versa for the Top flag. */

myAlignment = myAlignment XOR (Alignment:Top OR Alignmen:Bottom).
/* myAlignmnet now starts with the Left and Top flags set, and the enum
   instance created by the OR operation has both the Top and Bottom
   flags set. After the XOR operation, myAlignment has the Left and
   Bottom flags set because the status of those flags are different in
   the two enum instances operated on by the XOR.  */
```

The first XOR operation in the previous example shows how you can use bitwise XOR to toggle a flag on and off. Performing an XOR operation on `myAlignment` with an enum instance that has only one flag set will result in an instance with that flag turned on if if it was off in myAlignment or off it was turned on in `myAlignment`.

The second XOR operation shows how you can use bitwise XOR to toggle the status of two flags that you want to be mutually exclusive. When `myAlignment` starts with either `Top` or `Bottom` set, but not both, using the XOR operation with the enum instance created by the OR operation switches the status of the two flags.

Notes

- This operator can be used with both ABL and .NET flag enums.

See also

AND operator (bitwise), ENUM statement, NOT operator (bitwise), OR operator (bitwise)

YEAR function

Evaluates a date expression and returns the year value of that date, including the century, as an INTEGER value.

Syntax

```
YEAR ( date )
```

```
YEAR ( datetime-expression )
```

date

A date expression for which you want to determine the year.

datetime-expression

An expression that evaluates to a DATETIME or DATETIME-TZ. The YEAR function returns the year of the date part of the DATETIME or DATETIME-TZ value.

Example

This procedure uses the YEAR function to determine if an Order date is in this century or the next, and then uses a different display format for each:

r-year.p

```
DEFINE VARIABLE outfmt  AS CHARACTER NO-UNDO.
DEFINE VARIABLE orddate AS CHARACTER NO-UNDO
  LABEL "Order Date" FORMAT "x(10)".

FOR EACH Order NO-LOCK:
  ASSIGN
    outfmt  = IF YEAR(Order.OrderDate) >= 2000 THEN
                "99/99/9999" ELSE "99/99/99"
    orddate = STRING(Order.OrderDate, outfmt).

  DISPLAY Order.OrderNum orddate Order.Terms.
END.
```

See also

ADD-INTERVAL function, DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, ETIME function, INTERVAL function, ISO-DATE function, MONTH function, MTIME function, NOW function, TIME function, TIME-SOURCE attribute, TIMEZONE function, TODAY function, YEAR-OFFSET attribute

4

Widget Reference

This section contains reference entries that describe the ABL widgets. Widgets are handle-based objects that provide visual representations of data and other elements of a user interface.

Like all handle-based objects, widgets support attributes and methods to access and manipulate widget state and behavior. Widgets also support events to which an application can respond to interact with the state and behavior of each widget. For more information on the attributes and methods listed for each widget, see the Handle Attributes and Methods Reference. For more information on the events listed for each widget, see the Handle-based Object Events Reference. For information on non-visual handle-based objects, see the Handle Reference.

You may consider a user-interface widget to be supported for all interfaces and on all operating systems unless otherwise indicated in the reference entry. These user-interface widgets do not apply to SpeedScript programming.

Because widgets are not realized in batch mode, you cannot use any method or attribute that requires the widget to be realized in batch mode.

Note: Of the common attributes listed for the following widgets, BGCOLOR, FGCOLOR, FONT, MOVABLE, RESIZABLE, and SELECTABLE apply only to graphical interfaces; DCOLOR and PFCOLOR apply only to character interfaces. In character interfaces, all attributes and methods that reference pixels (for example HEIGHT-PIXELS) use a system default pixel value for the equivalent value in characters.

The following descriptions refer to both compile-time and run-time behavior, features that the language generally supports and determines at compile time and actions directed by using these features at run time. When describing compile-time features or actions, this section references ABL or the ABL compiler. When describing ABL-directed actions taken at run time, this section references the ABL Virtual Machine (AVM).

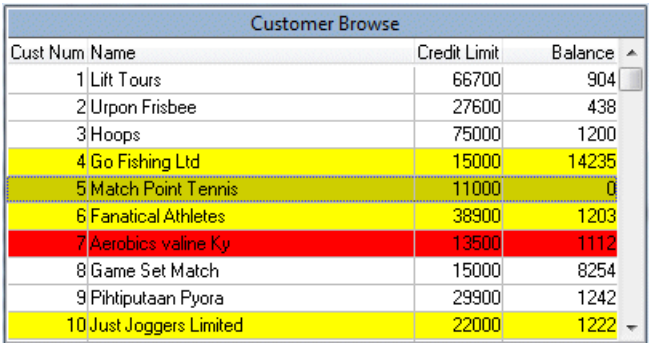
BROWSE widget

A browse widget lets you see data and select records from all the records associated with a database query. You can define a static browse widget with the DEFINE BROWSE statement or a dynamic browse widget with the CREATE BROWSE statement. A browse can be either a read-only tool for browsing through records, or it can be an editing tool for updating records, depending on the options you specify.

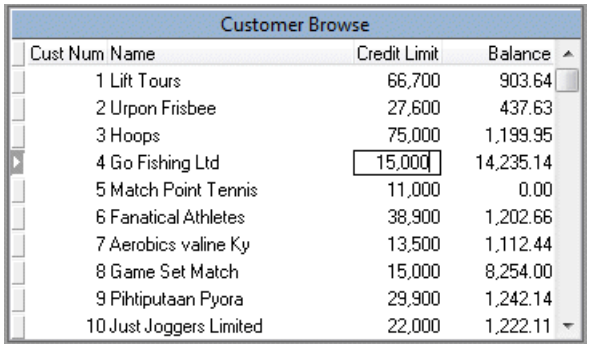
You can move and resize the browse and its components. Specifically, in graphical interfaces, you can move and resize the browse, move and change the width of the browse-column, and change the height of the browse-row. You can do all this through direct manipulation (by pointing, clicking, and dragging) and through ABL.

You can also use the mouse wheel to scroll the browse widget horizontally and vertically in Windows. When you rotate the mouse wheel up and down, the browse scrolls vertically up and down. When you rotate the mouse wheel up and down while holding down the **CTRL** key, the browse scrolls horizontally left and right. You can specify the number of rows the browse scrolls up and down per click of the mouse wheel on the Wheel tab in the Windows Mouse Properties dialog box (accessed through the Windows Control Panel).

The following figure shows a read-only browse widget with colors applied to some of the rows using the BGCOLOR attribute. Note that the fifth row is selected:



The following figure shows an updatable browse. Note the inline editing capability in the focused row:



Accessing browse attributes and methods

When accessing browse attributes and methods, it is important to understand the scope of each attribute and method. An attribute or method can apply to:

- The browse widget as a whole.
- A single browse column. In this case, any attribute or method that applies to a browse column, applies to:
  - All cells in the browse column (that is, all cells of the given column for all rows of the browse)
  - Any type of browse column (fill-in, combo-box, or toggle-box), except where noted

- A single browse cell. In this case, the attribute or method applies to only the given cell at the intersection of the referenced column and the focused row.
- Both the browse as a whole and a column or cell. For example, in the same trigger, you could change the background color of the whole browse to blue and the background color of the current cell to yellow.

Syntax

When accessing attributes and methods that apply to a browse widget as a whole, you must reference the browse widget using its name or handle, as shown in the following syntax examples:

```
/* For a static browse */
browse-name:attribute-or-method-name IN FRAME frame-name

/* For a dynamic or static browse */
browse-handle:attribute-or-method-name
```

The IN FRAME qualifier is only necessary for a static browse to avoid ambiguity.

When accessing attributes and methods that apply to a browse column or cell, you must reference the browse column or cell using the browse column's name or handle, as shown in the following syntax examples:

```
/* For a static browse column */
column-name:attribute-or-method-name IN BROWSE browse-name

/* For a dynamic or static browse column */
column-handle:attribute-or-method-name
```

The IN BROWSE qualifier is only necessary for a static browse to avoid ambiguity, but it is good programming practice to always include it, especially when you reference the same field as a separate widget type.

To access attributes and methods for a specific browse cell, you must be sure that a row is selected. You typically access browse cell attributes and methods in a ROW-DISPLAY trigger.

For more information about accessing widget attributes and methods, see the Handle Attributes and Methods Reference.

Attributes

The following table lists all the attributes for the browse widget and their scope:

Attribute	Applies to
ALLOW-COLUMN-SEARCHING attribute	Browse
AUTO-COMPLETION attribute[61]	Column
AUTO-RESIZE attribute	Column
AUTO-RETURN attribute	Column
AUTO-VALIDATE attribute	Browse
AUTO-ZAP attribute [62]	Cell

Attribute	Applies to
BGCOLOR attribute	Browse, Column, Cell
BUFFER-FIELD attribute	Column
COLUMN attribute	Browse, Column
COLUMN-BGCOLOR attribute	Column
COLUMN-DCOLOR attribute	Column
COLUMN-FGCOLOR attribute	Column
COLUMN-FONT attribute	Column
COLUMN-MOVABLE attribute	Browse
COLUMN-PFCOLOR attribute	Column
COLUMN-READ-ONLY attribute	Column
COLUMN-RESIZABLE attribute	Browse
COLUMN-SCROLLING attribute	Browse
CONTEXT-HELP-ID attribute	Browse
CURRENT-COLUMN attribute	Browse
CURRENT-ROW-MODIFIED attribute	Browse
CURSOR-OFFSET attribute[62]	Cell
DATA-TYPE attribute	Column
DBNAME attribute	Column
DCOLOR attribute	Browse, Column, Cell
DELIMITER attribute[61]	Column
DISABLE-AUTO-ZAP attribute	Column
DOWN attribute	Browse
DROP-TARGET attribute	Browse
DYNAMIC attribute	Browse
EDIT-CAN-PASTE attribute[62]	Cell
EDIT-CAN-UNDO attribute[62]	Cell

Attribute	Applies to
EXPANDABLE attribute	Browse
FGCOLOR attribute	Browse, Column, Cell
FIRST-COLUMN attribute	Browse
FIT-LAST-COLUMN attribute	Browse
FOCUSED-ROW attribute	Browse
FOCUSED-ROW-SELECTED attribute	Browse
FONT attribute[62]	Browse, Column, Cell
FORMAT attribute[62]	Cell
FRAME attribute	Browse
FRAME-COL attribute	Browse
FRAME-NAME attribute	Browse
FRAME-ROW attribute	Browse
FRAME-X attribute	Browse
FRAME-Y attribute	Browse
HANDLE attribute	Browse, Column, Cell
HEIGHT-CHARS attribute	Browse, Cell
HEIGHT-PIXELS attribute	Browse, Cell
HELP attribute	Browse, Column
HIDDEN attribute	Browse
HWND attribute	Browse, Cell
INDEX attribute	Column, Cell
INNER-LINES attribute[61]	Column
INPUT-VALUE attribute	Cell
INSTANTIATING-PROCEDURE attribute	Browse, Column
LABEL attribute	Column
LABELS attribute	Browse

Attribute	Applies to
LABEL-BGCOLOR attribute	Browse, Column
LABEL-DCOLOR attribute	Browse, Column
LABEL-FGCOLOR attribute	Browse, Column
LABEL-FONT attribute	Browse, Column
LIST-ITEM-PAIRS attribute[61]	Column
LIST-ITEMS attribute[61]	Column
MANUAL-HIGHLIGHT attribute	Browse
MAX-CHARS attribute[61]	Column
MAX-DATA-GUESS attribute	Browse
MENU-KEY attribute	Browse
MENU-MOUSE attribute	Browse
MIN-COLUMN-WIDTH-CHARS attribute	Browse
MIN-COLUMN-WIDTH-PIXELS attribute	Browse
MIN-HEIGHT-CHARS attribute	Browse
MODIFIED attribute	Browse, Column
MOUSE-POINTER attribute	Browse, Column
MOVABLE attribute	Browse, Column
MULTIPLE attribute	Browse
NAME attribute	Browse, Column, Cell
NEW-ROW attribute	Browse
NEXT-COLUMN attribute	Column
NEXT-SIBLING attribute	Browse
NEXT-TAB-ITEM attribute	Browse
NO-EMPTY-SPACE attribute	Browse
NO-VALIDATE attribute	Browse
NUM-COLUMNS attribute	Browse

Attribute	Applies to
NUM-DROPPED-FILES attribute	Browse
NUM-ITEMS attribute[61]	Column
NUM-ITERATIONS attribute (widget objects)	Browse
NUM-LOCKED-COLUMNS attribute	Browse
NUM-SELECTED-ROWS attribute	Browse
NUM-VISIBLE-COLUMNS attribute	Browse
PARENT attribute	Browse, Column
PFCOLOR attribute	Browse, Column, Cell
POPUP-MENU attribute	Browse
PREV-COLUMN attribute	Column
PREV-SIBLING attribute	Browse
PREV-TAB-ITEM attribute	Browse
PRIVATE-DATA attribute	Browse, column
QUERY attribute	Browse
READ-ONLY attribute	Browse, column
REFRESHABLE attribute	Browse
RESIZABLE attribute	Browse, column
ROW attribute	Browse, cell
ROW-HEIGHT-CHARS attribute	Browse
ROW-HEIGHT-PIXELS attribute	Browse
ROW-MARKERS attribute	Browse
ROW-RESIZABLE attribute	Browse
SCREEN-VALUE attribute	Cell
SCROLLBAR-VERTICAL attribute	Browse
SELECTABLE attribute	Browse
SELECTED attribute	Browse

Attribute	Applies to
SELECTION-END attribute	Column
SELECTION-START attribute	Column
SELECTION-TEXT attribute	Column
SENSITIVE attribute	Browse
SEPARATORS attribute	Browse
SEPARATOR-FGCOLOR attribute	Browse
SORT attribute[61]	Column
SORT-ASCENDING attribute	Column
SORT-NUMBER attribute	Column
SUBTYPE attribute[61]	Column
TAB-POSITION attribute	Browse
TAB-STOP attribute	Browse
TABLE attribute	Browse, Column
TEXT-SELECTED attribute	Column
TITLE attribute	Browse
TITLE-BGCOLOR attribute	Browse
TITLE-DCOLOR attribute	Browse
TITLE-FGCOLOR attribute	Browse
TITLE-FONT attribute	Browse
TOOLTIP attribute	Browse
TYPE attribute	Browse, Column, Cell
UNIQUE-MATCH attribute[61]	Column
VIEW-FIRST-COLUMN-ON-REOPEN attribute	Browse
VISIBLE attribute	Browse, Column
WIDGET-ID attribute	Browse
WIDTH-CHARS attribute	Browse, Column, Cell

Attribute	Applies to
WIDTH-PIXELS attribute	Browse, Column, Cell
WINDOW attribute	Browse
X attribute	Browse, Column
Y attribute	Browse, Cell

Methods

The following table lists all the methods for the browse widget and their scope:

Method	Applies to
ADD-CALC-COLUMN( ) method	Browse
ADD-COLUMNS-FROM( ) method	Browse
ADD-FIRST( ) method[63]	Column
ADD-LAST( ) method[63]	Column
ADD-LIKE-COLUMN( ) method	Browse
CLEAR-SELECTION( ) method[64]	Column
CLEAR-SORT-ARROWS( ) method	Browse
CREATE-RESULT-LIST-ENTRY( ) method	Browse
DELETE( ) method[63]	Column
DELETE-CURRENT-ROW( ) method	Browse
DELETE-RESULT-LIST-ENTRY( ) method	Browse
DELETE-SELECTED-ROW( ) method	Browse
DELETE-SELECTED-ROWS( ) method	Browse
DESELECT-FOCUSED-ROW( ) method	Browse
DESELECT-ROWS( ) method	Browse
DESELECT-SELECTED-ROW( ) method	Browse
EDIT-CLEAR( ) method[64]	Cell

[61] This attribute applies only to combo-box browse columns.
[62] This attribute applies only to fill-in and combo-box browse columns.

Method	Applies to
EDIT-COPY( ) method[64]	Cell
EDIT-CUT( ) method[64]	Cell
EDIT-PASTE( ) method[64]	Cell
EDIT-UNDO( ) method[64]	Cell
END-FILE-DROP( ) method	Browse
ENTRY( ) method[63]	Column
FETCH-SELECTED-ROW( ) method	Browse
GET-BROWSE-COLUMN( ) method	Browse
GET-DROPPED-FILE( ) method	Browse
GET-REPOSITIONED-ROW( ) method	Browse
INSERT( ) method[63]	Column
INSERT-ROW( ) method	Browse
IS-ROW-SELECTED( ) method	Browse
LOAD-MOUSE-POINTER( ) method	Browse, Column
LOOKUP( ) method[63]	Column
MOVE-AFTER-TAB-ITEM( ) method	Browse
MOVE-BEFORE-TAB-ITEM( ) method	Browse
MOVE-COLUMN( ) method	Browse
MOVE-TO-BOTTOM( ) method	Browse
MOVE-TO-TOP( ) method	Browse
REFRESH( ) method (Handle)	Browse
REPLACE( ) method[63]	Column
SCROLL-TO-CURRENT-ROW( ) method	Browse
SCROLL-TO-SELECTED-ROW( ) method	Browse
SELECT-ALL( ) method	Browse
SELECT-FOCUSED-ROW( ) method	Browse

Method	Applies to
SELECT-NEXT-ROW( ) method	Browse
SELECT-PREV-ROW( ) method	Browse
SELECT-ROW( ) method	Browse
SET-REPOSITIONED-ROW( ) method	Browse
SET-SELECTION( ) method[64]	Column
SET-SORT-ARROW( ) method	Browse
VALIDATE( ) method	Browse, Column

Events

The following table lists all the events for the browse widget and their scope:

Event	Applies to
Default keyboard events	Browse, Cell
Developer events	Browse, Column, Cell
Field editing key function events	Cell
General direct manipulation events	Browse, Column
Mouse events	Browse, Column, Cell
Navigation key function events	Browse, Cell
Universal key function events	Browse, Column, Cell
DEFAULT-ACTION	Browse
DROP-FILE-NOTIFY	Browse
END	Browse
END-SEARCH	Browse
ENTRY	Browse, Cell
HOME	Browse
LEAVE	Browse, Cell

[63] This method applies only to combo-box browse columns.
[64] This method applies only to fill-in and combo-box browse columns.

Event	Applies to
OFF-END	Browse
OFF-HOME	Browse
ROW-DISPLAY	Browse
ROW-ENTRY	Browse
ROW-LEAVE	Browse
SCROLL-NOTIFY	Browse
START-SEARCH	Browse
VALUE-CHANGED	Browse

See also

DEFINE BROWSE statement, CREATE BROWSE statement

BUTTON widget

A button widget represents a push button on the screen. The button can contain a textual label or it can have images associated with its pressed and unpressed states. You can define a static button with the DEFINE BUTTON statement. You can create dynamic buttons with the CREATE widget statement. This figure shows three buttons:



Attributes

AUTO-END-KEY attribute	AUTO-GO attribute	AUTO-RESIZE attribute
COLUMN attribute	CONTEXT-HELP-ID attribute	CONVERT-3D-COLORS attribute
DCOLOR attribute	DEFAULT attribute	DROP-TARGET attribute
DYNAMIC attribute	FLAT-BUTTON attribute	FONT attribute
FRAME attribute	FRAME-COL attribute	FRAME-NAME attribute
FRAME-ROW attribute	FRAME-X attribute	FRAME-Y attribute
HANDLE attribute	HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute
HELP attribute	HIDDEN attribute	HTML-CHARSET attribute
IMAGE attribute	IMAGE-DOWN attribute	IMAGE-INSENSITIVE attribute

IMAGE-UP attribute	INSTANTIATING-PROCEDURE attribute	LABEL attribute
MANUAL-HIGHLIGHT attribute	MENU-KEY attribute	MENU-MOUSE attribute
MOUSE-POINTER attribute	MOVABLE attribute	NAME attribute
NEXT-SIBLING attribute	NEXT-TAB-ITEM attribute	NO-FOCUS attribute
NUM-DROPPED-FILES attribute	PARENT attribute	PFCOLOR attribute
POPUP-MENU attribute	PREV-SIBLING attribute	PREV-TAB-ITEM attribute
PRIVATE-DATA attribute	RESIZABLE attribute	ROW attribute
SELECTABLE attribute	SELECTED attribute	SENSITIVE attribute
TAB-POSITION attribute	TAB-STOP attribute	TOOLTIP attribute
TYPE attribute	VISIBLE attribute	WIDGET-ID attribute
WIDTH-CHARS attribute	WIDTH-PIXELS attribute	WINDOW attribute
X attribute	Y attribute	–

Methods

END-FILE-DROP( ) method	GET-DROPPED-FILE( ) method
LOAD-IMAGE( ) method	LOAD-IMAGE-DOWN( ) method
LOAD-IMAGE-INSENSITIVE( ) method	LOAD-IMAGE-UP( ) method
LOAD-MOUSE-POINTER( ) method	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	–

Events

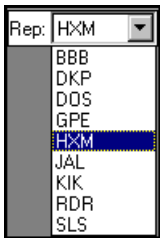
Default keyboard events	Developer events
Mouse events	Navigation key function events
Universal key function events	CHOOSE
DROP-FILE-NOTIFY	ENTRY
LEAVE	–

See also

DEFINE BUTTON statement, CREATE widget statement

COMBO-BOX widget

A combo box is a field-level widget that combines the functionality of a fill-in field, radio set, and selection list into one fill-in and drop down list. You can set up a static combo box widget with the VIEW-AS phrase. You can create a dynamic combo box with the CREATE widget statement. The following figure shows a combo box:



Attributes

AUTO-COMPLETION attribute[65]	AUTO-RESIZE attribute	AUTO-ZAP attribute[65]
BGCOLOR attribute[65]	COLUMN attribute	CONTEXT-HELP-ID attribute
DATA-TYPE attribute	DBNAME attribute	DCOLOR attribute[65]
DELIMITER attribute[65]	DISABLE-AUTO-ZAP attribute	DROP-TARGET attribute
DYNAMIC attribute	EDIT-CAN-PASTE attribute[65]	EDIT-CAN-UNDO attribute[65]
FGCOLOR attribute[65]	FONT attribute[65]	FORMAT attribute[65]
FRAME attribute	FRAME-COL attribute	FRAME-NAME attribute
FRAME-ROW attribute	FRAME-X attribute	FRAME-Y attribute
HANDLE attribute[65]	HEIGHT-CHARS attribute[65]	HEIGHT-PIXELS attribute[65]
HELP attribute	HIDDEN attribute	HWND attribute
INDEX attribute	INNER-LINES attribute[65]	INPUT-VALUE attribute[65]
INSTANTIATING-PROCEDURE attribute	LABEL attribute	LABELS attribute
LIST-ITEM-PAIRS attribute[65]	LIST-ITEMS attribute[65]	MANUAL-HIGHLIGHT attribute
MAX-CHARS attribute[65]	MENU-KEY attribute	MENU-MOUSE attribute

[65] This attribute also applies to combo-box browse columns.

MODIFIED attribute	MOUSE-POINTER attribute	MOVABLE attribute
NAME attribute[65]	NEXT-SIBLING attribute	NEXT-TAB-ITEM attribute
NUM-DROPPED-FILES attribute	NUM-ITEMS attribute[65]	PARENT attribute
PFCOLOR attribute[65]	POPUP-MENU attribute	PREV-SIBLING attribute
PREV-TAB-ITEM attribute	PRIVATE-DATA attribute	RESIZABLE attribute
ROW attribute[65]	SCREEN-VALUE attribute[65]	SELECTABLE attribute
SELECTED attribute	SELECTION-END attribute	SELECTION-START attribute
SELECTION-TEXT attribute	SENSITIVE attribute	SIDE-LABEL-HANDLE attribute
SORT attribute[65]	SUBTYPE attribute[65]	TABLE attribute
TAB-POSITION attribute	TAB-STOP attribute	TEXT-SELECTED attribute
TOOLTIP attribute	TYPE attribute[65]	UNIQUE-MATCH attribute[65]
VISIBLE attribute	WIDGET-ID attribute	WIDTH-CHARS attribute[65]
WIDTH-PIXELS attribute[65]	WINDOW attribute	X attribute[65]
Y attribute[65]	–	–

Methods

ADD-FIRST( ) method[66]	ADD-LAST( ) method[66]
CLEAR-SELECTION( ) method[66]	DELETE( ) method[66]
EDIT-CLEAR( ) method[66]	EDIT-COPY( ) method[66]
EDIT-CUT( ) method[66]	EDIT-PASTE( ) method[66]
EDIT-UNDO( ) method[66]	END-FILE-DROP( ) method
ENTRY( ) method[66]	GET-DROPPED-FILE( ) method
INSERT( ) method[66]	LOAD-MOUSE-POINTER( ) method
LOOKUP( ) method[66]	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-TO-BOTTOM( ) method

[66] This method also applies to combo-box browse columns.

MOVE-TO-TOP( ) method	REPLACE( ) method[66]
SET-SELECTION( ) method[66]	VALIDATE( ) method

Events

Default keyboard events	Developer events
Field editing key function events	General direct manipulation events
Mouse events	Navigation key function events
Universal key function events	DROP-FILE-NOTIFY
ENTRY	LEAVE
VALUE-CHANGED	–

See also

CREATE widget statement, VIEW-AS phrase

CONTROL-FRAME widget
(Windows only; Graphical interfaces only)

A control-frame is a field-level widget that holds an ActiveX control that you select for your application from the OpenEdge AppBuilder. A control-frame is always created dynamically.

A control-frame has no visualization.

The AVM instantiates two separate but related objects when you create a control-frame:

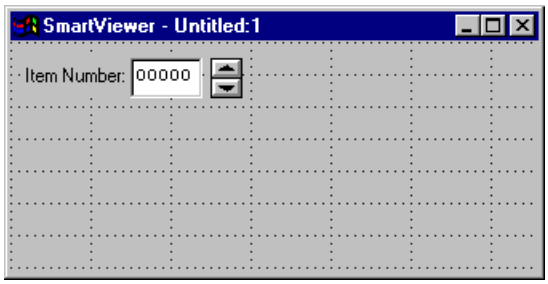
- A control-frame widget
- A control-frame COM object

The widget itself provides a connection between the ActiveX control and the ABL user interface. When the widget is realized, the AVM creates a COM object that provides the real ActiveX control container support. Thus, the control-frame widget provides widget attributes and methods to manage the ABL side of the interface, while the control-frame COM object provides COM object properties and methods to gain access to the control itself.

When you insert an ActiveX control into your application, the AppBuilder creates a control-frame with the CREATE widget statement and specifies a default name (NAME attribute value) for the widget. The AppBuilder creates a design-time instance of the ActiveX control based on the control you select in the AppBuilder, making its design-time properties available to the AppBuilder. When you save your application, the AppBuilder saves the design-time instance in a separate file (with `.wrx` extension) for use at run time.

At run time, your application accesses the control indirectly through the control-frame widget. First, you use the COM-HANDLE attribute to return a component handle to the control-frame COM object. Second, you use this handle to access properties and methods of the control-frame COM object, which provide access to the ActiveX control itself.

This is a SmartViewer into which a developer, using the AppBuilder, has dropped a literal widget, a fill-in widget, and a control-frame widget. The control-frame widget holds a Crescent spin control, as shown:



Attributes

Control-frame widget attributes

BGCOLOR attribute	COLUMN attribute	COM-HANDLE attribute
CONTEXT-HELP-ID attribute	DYNAMIC attribute	FRAME attribute
FRAME-COL attribute	FRAME-NAME attribute	FRAME-ROW attribute
FRAME-X attribute	FRAME-Y attribute	HEIGHT-CHARS attribute
HEIGHT-PIXELS attribute	HELP attribute	HIDDEN attribute
INSTANTIATING-PROCEDURE attribute	HTML-CHARSET attribute	NAME attribute
NEXT-SIBLING attribute	NEXT-TAB-ITEM attribute	PARENT attribute
PREV-SIBLING attribute	PREV-TAB-ITEM attribute	PRIVATE-DATA attribute
ROW attribute	SENSITIVE attribute	TAB-POSITION attribute
TAB-STOP attribute	TYPE attribute	VISIBLE attribute
WIDGET-ID attribute	WIDTH-CHARS attribute	WIDTH-PIXELS attribute
WINDOW attribute	X attribute	Y attribute

Properties

Control-frame COM object properties[67]

Control-Name property [68]	Controls property	Height property[69]
Left property[69]	Name property[69]	Top property[69]
Widget-Handle property	Width property[69]	–

[67] Accessible using a component handle set to the control-frame COM-HANDLE attribute value.The name of an ActiveX control that is contained by the control-frame COM object.Mapped to a corresponding control-frame widget attribute.

[68] The name of an ActiveX control that is contained by the control-frame COM object

[69] Mapped to a corresponding control-frame widget attribute.

Methods

Control-frame widget methods:

ADD-EVENTS-PROCEDURE( ) method	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	REMOVE-EVENTS-PROCEDURE( ) method

Control-frame COM object methods:[70]

LoadControls( ) method	–

Events

Developer events	BACK-TAB navigation key function event
END-ERROR universal key function event	ENTRY
GO universal key function event	HELP universal key function event
LEAVE	TAB navigation key function event

Notes

- You must use the AppBuilder to incorporate one or more ActiveX control instances into an ABL application. The AppBuilder, operating in design mode, provides the facilities to set design-time properties for ActiveX controls.
- After incorporating ActiveX controls into an application with the AppBuilder, the resulting window file, when compiled and executed, interacts with the ActiveX controls at run time.
- To access a loaded ActiveX control at run time, use the control-frame COM-HANDLE attribute to get a handle to the control-frame COM object. To return a handle to the control, use the design-time name of the ActiveX control as a property of the control-frame COM object. For example:

```
/* Control Frame widget */
DEFINE VARIABLE hCFwid    AS HANDLE     NO-UNDO.
/* Control Frame COM Object */
DEFINE VARIABLE hCFcom    AS COM-HANDLE NO-UNDO.
/* ActiveX Control */
DEFINE VARIABLE hDateSpin AS COM-HANDLE NO-UNDO.

/* Control-frame created with handle hCFwid and loaded with ActiveX
   control named DateSpin. */

ASSIGN
  hCFcom    = hCFwid:COM-HANDLE
  hDateSpin = hCFcom:DateSpin.
```

[70] Accessible using a component handle set to the control-frame COM-HANDLE attribute value.

As an alternative, use the COM object Controls property to return a handle to a control collection. Use the control collection Item(1) method call to return the handle to the ActiveX control. (This control collection object provides support for searching multiple ActiveX controls in a control-frame, available in a future release of OpenEdge.)

- You can use a single ActiveX control more than once in a single window file. Each time you insert the control, the AppBuilder creates a separate control-frame for it with a unique NAME attribute value.
- Some control-frame widget attributes correspond to control-frame COM object properties so that setting one sets the other. You must directly set and read all ActiveX control run-time properties using a handle (also a COM-HANDLE value) to the control.
- To trap control-frame events, use the ON statement, as with any ABL widget. To trap events for the associated ActiveX control, you must use ActiveX control (OCX) event procedures. Also, to "apply" an ActiveX control event from ABL, run the event procedure directly, like any ABL internal procedure. The APPLY statement has no effect on ActiveX controls. For more information, see the reference entries for the PROCEDURE statement and RUN statement.
- ABL control-frame events are mutually exclusive with associated ActiveX control events. That is, only one event handler, either an ON trigger or an event procedure, fires for a single event.

See also

CREATE widget statement, Topics on ActiveX control container support in *OpenEdge Programming Interfaces*

DIALOG-BOX widget

A dialog box is a special type of frame that is displayed in its own window. A dialog box differs from a window in two major respects:

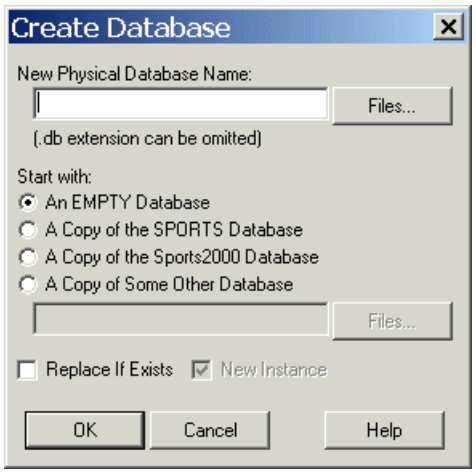
- It has a system window ventilator, but has no affordances for minimizing or maximizing.
- While a dialog box has input focus, your application cannot perform any other processing until you complete the input or otherwise close the dialog box. That is, it is modal.

You can specify that a frame be displayed as a dialog box by using the VIEW-AS phrase. You can create a dynamic dialog box with the CREATE widget statement.

A dialog box can contain a frame family acting as the root frame. However a dialog box cannot be a child of another frame or dialog box; it can only be parented by a window.

The following example dialog box contains:

- Two fill-ins
- One radio set
- One toggle box
- Five buttons



Attributes

BACKGROUND attribute	BGCOLOR attribute	BORDER-BOTTOM-CHARS attribute
BORDER-BOTTOM-PIXELS attribute	BORDER-LEFT-CHARS attribute	BORDER-LEFT-PIXELS attribute
BORDER-RIGHT-CHARS attribute	BORDER-RIGHT-PIXELS attribute	BORDER-TOP-CHARS attribute
BORDER-TOP-PIXELS attribute	BOX-SELECTABLE attribute	CANCEL-BUTTON attribute
COLUMN attribute	CONTEXT-HELP attribute	CONTEXT-HELP-FILE attribute
CURRENT-ITERATION attribute (Widget Objects)	DCOLOR attribute	DEFAULT-BUTTON attribute
DROP-TARGET attribute	DYNAMIC attribute	FGCOLOR attribute
FIRST-CHILD attribute	FONT attribute	HANDLE attribute
HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute	HIDDEN attribute
HTML-CHARSET attribute	INHERIT-BGCOLOR attribute	INHERIT-FGCOLOR attribute
INSTANTIATING-PROCEDURE attribute	LAST-CHILD attribute	MENU-KEY attribute
MENU-MOUSE attribute	MOUSE-POINTER attribute	NAME attribute
NEXT-SIBLING attribute	NUM-DROPPED-FILES attribute	NUM-SELECTED-WIDGETS attribute
PARENT attribute	PFCOLOR attribute	POPUP-MENU attribute
PREV-SIBLING attribute	PRIVATE-DATA attribute	ROW attribute
SCROLLABLE attribute	SENSITIVE attribute	THREE-D attribute

TITLE attribute	TITLE-BGCOLOR attribute	TITLE-DCOLOR attribute
TITLE-FGCOLOR attribute	TITLE-FONT attribute	TYPE attribute
VIRTUAL-HEIGHT-CHARS attribute	VIRTUAL-HEIGHT-PIXELS attribute	VIRTUAL-WIDTH-CHARS attribute
VIRTUAL-WIDTH-PIXELS attribute	VISIBLE attribute	WIDGET-ID attribute
WIDTH-CHARS attribute	WIDTH-PIXELS attribute	WINDOW attribute
X attribute	Y attribute	–

Methods

END-FILE-DROP( ) method	GET-DROPPED-FILE( ) method)
GET-SELECTED-WIDGET( ) method	LOAD-MOUSE-POINTER( ) method
VALIDATE( ) method	–

Events

Developer events	Frame-only direct manipulation events
Universal key function events	DROP-FILE-NOTIFY
ENTRY	LEAVE
WINDOW-CLOSE	–

Note

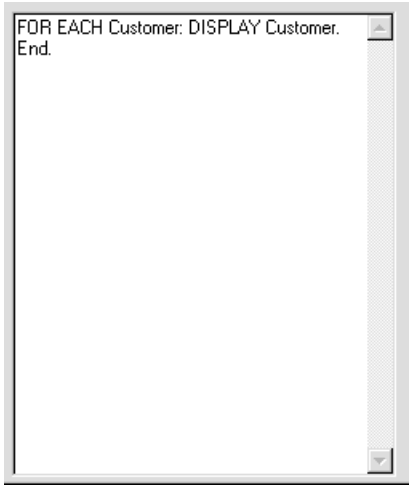
Generally, your application must wait to complete dialog box input before continuing with other processing. However, the WAIT-FOR statement for the procedure can also respond to an event for a procedure handle as long as the widget in the WAIT-FOR statement widget list is a procedure handle.

See also

CREATE widget statement, FRAME widget, WAIT-FOR statement (ABL only), VIEW-AS phrase, WINDOW widget

EDITOR widget

An editor is a field-level widget that allows you to perform complex text manipulation on a character value. You can set up a static editor widget with the VIEW-AS phrase. You can create a dynamic editor widget with the CREATE widget statement. For example:



Attributes

AUTO-INDENT attribute	AUTO-RESIZE attribute	BGCOLOR attribute
BOX attribute	BUFFER-CHARS attribute	BUFFER-LINES attribute
COLUMN attribute	CONTEXT-HELP-ID attribute	CURSOR-CHAR attribute
CURSOR-LINE attribute	CURSOR-OFFSET attribute	DATA-TYPE attribute
DBNAME attribute	DCOLOR attribute	DROP-TARGET attribute
DYNAMIC attribute	EDIT-CAN-PASTE attribute	EDIT-CAN-UNDO attribute
EMPTY attribute	FGCOLOR attribute	FONT attribute
FRAME attribute	FRAME-COL attribute	FRAME-NAME attribute
FRAME-ROW attribute	FRAME-X attribute	FRAME-Y attribute
HANDLE attribute	HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute
HELP attribute	HIDDEN attribute	HTML-CHARSET attribute
INNER-CHARS attribute	INNER-LINES attribute	INPUT-VALUE attribute
INSTANTIATING-PROCEDURE attribute	LABEL attribute	LABELS attribute
LARGE attribute	LENGTH attribute	MANUAL-HIGHLIGHT attribute
MAX-CHARS attribute	MENU-KEY attribute	MENU-MOUSE attribute
MODIFIED attribute	MOUSE-POINTER attribute	MOVABLE attribute
NAME attribute	NEXT-SIBLING attribute	NEXT-TAB-ITEM attribute
NUM-DROPPED-FILES attribute	NUM-LINES attribute	NUM-REPLACED attribute

PARENT attribute	PFCOLOR attribute	POPUP-MENU attribute
PREV-SIBLING attribute	PREV-TAB-ITEM attribute	PRIVATE-DATA attribute
PROGRESS-SOURCE attribute	READ-ONLY attribute	RESIZABLE attribute
RETURN-INSERTED attribute	ROW attribute	SCREEN-VALUE attribute
SCROLLBAR-HORIZONTAL attribute	SCROLLBAR-VERTICAL attribute	SELECTABLE attribute
SELECTED attribute	SELECTION-END attribute	SELECTION-START attribute
SELECTION-TEXT attribute	SENSITIVE attribute	SIDE-LABEL-HANDLE attribute
TABLE attribute	TAB-POSITION attribute	TAB-STOP attribute
TEXT-SELECTED attribute	TOOLTIP attribute	TYPE attribute
VISIBLE attribute	WIDGET-ID attribute	WIDTH-CHARS attribute
WIDTH-PIXELS attribute	WINDOW attribute	WORD-WRAP attribute
X attribute	Y attribute	–

Methods

CLEAR-SELECTION( ) method	CONVERT-TO-OFFSET( ) method
DELETE-CHAR( ) method	DELETE-LINE( ) method
EDIT-CLEAR( ) method	EDIT-COPY( ) method
EDIT-CUT( ) method	EDIT-PASTE( ) method
EDIT-UNDO( ) method	END-FILE-DROP( ) method
GET-DROPPED-FILE( ) method	INSERT-BACKTAB( ) method
INSERT-FILE( ) method	INSERT-STRING( ) method
INSERT-TAB( ) method	LOAD-MOUSE-POINTER( ) method
MOVE-AFTER-TAB-ITEM( ) method	MOVE-BEFORE-TAB-ITEM( ) method
MOVE-TO-BOTTOM( ) method	MOVE-TO-EOF( ) method
MOVE-TO-TOP( ) method	READ-FILE( ) method
REPLACE( ) method	REPLACE-SELECTION-TEXT( ) method

SAVE-FILE( ) method	SEARCH( ) method
SET-SELECTION( ) method	VALIDATE( ) method

Events

Default keyboard events	Developer events
General direct manipulation events	Mouse events
Navigation key function events	Universal key function events
DROP-FILE-NOTIFY	ENTRY
LEAVE	VALUE-CHANGED

See also

CREATE widget statement, VIEW-AS phrase

FIELD-GROUP widget

A field group is the hidden parent of field-level widgets and child frames owned by a parent frame or dialog box. Thus, field groups are the actual children of frames and dialog boxes. A frame contains the following field groups:

- A background field group (which includes the frame header)
- For a one-down frame or dialog box: a single data field group containing field-level widgets and child frames
- For a multiple-down frame: one data field group for each data iteration in the frame

A field group has no visible representation. You cannot explicitly define or create field groups. They are generated automatically when frames are defined or created.

Attributes

COLUMN attribute	DYNAMIC attribute	FIRST-CHILD attribute
FIRST-TAB-ITEM attribute	FOREGROUND attribute	HANDLE attribute
HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute	HTML-CHARSET attribute
INSTANTIATING-PROCEDURE attribute	LAST-CHILD attribute	LAST-TAB-ITEM attribute
NAME attribute	NEXT-SIBLING attribute	NUM-TABS attribute
PARENT attribute	PREV-SIBLING attribute	PRIVATE-DATA attribute
ROW attribute	SENSITIVE attribute	TYPE attribute

VISIBLE attribute	WIDTH-CHARS attribute	WIDTH-PIXELS attribute
WINDOW attribute	X attribute	Y attribute

Note: For a field group, all of these attributes are read-only except for PRIVATE-DATA and SENSITIVE.

Methods

GET-TAB-ITEM( ) method	–

Events

The FIELD-GROUP widget does not support any events.

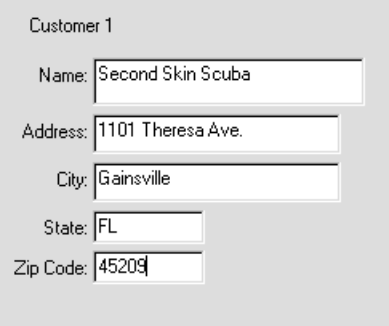
See also

DIALOG-BOX widget, FRAME widget

FILL-IN widget

A fill-in widget is the simplest form of data representation. Within a fill-in, the field value is displayed as a string of characters that you can edit. A fill-in is the default representation for data. You can explicitly set up a static fill-in with the VIEW-AS phrase. You can create a dynamic fill-in with the CREATE widget statement.

Note: The default sizing of fill-ins occurs only when you use the default font. When you explicitly specify a font, the AVM uses the average width of that font.



Attributes

ATTR-SPACE attribute	AUTO-RESIZE attribute	AUTO-RETURN attribute
AUTO-ZAP attribute[71]	BGCOLOR attribute[71]	BLANK attribute
COLUMN attribute	CONTEXT-HELP-ID attribute	CURSOR-OFFSET attribute[71]
DATA-TYPE attribute	DBNAME attribute	DCOLOR attribute[71]

[71] This attribute also applies to fill-in browse columns.

DEBLANK attribute	DISABLE-AUTO-ZAP attribute	DROP-TARGET attribute
DYNAMIC attribute	EDIT-CAN-PASTE attribute[71]	EDIT-CAN-UNDO attribute[71]
FGCOLOR attribute[71]	FONT attribute[71]	FORMAT attribute[71]
FRAME attribute	FRAME-COL attribute	FRAME-NAME attribute
FRAME-ROW attribute	FRAME-X attribute	FRAME-Y attribute
HANDLE attribute[71]	HEIGHT-CHARS attribute[71]	HEIGHT-PIXELS attribute[71]
HELP attribute	HIDDEN attribute	HWND attribute
INDEX attribute	INPUT-VALUE attribute[71]	INSTANTIATING-PROCEDURE attribute
LABEL attribute	LABELS attribute	MANUAL-HIGHLIGHT attribute
MENU-KEY attribute	MENU-MOUSE attribute	MODIFIED attribute
MOUSE-POINTER attribute	MOVABLE attribute	NAME attribute[71]
NEXT-SIBLING attribute	NEXT-TAB-ITEM attribute	NUM-DROPPED-FILES attribute
PARENT attribute	PASSWORD-FIELD attribute	PFCOLOR attribute[71]
POPUP-MENU attribute	PREV-SIBLING attribute	PREV-TAB-ITEM attribute
PRIVATE-DATA attribute	READ-ONLY attribute	RESIZABLE attribute
ROW attribute[71]	SCREEN-VALUE attribute[71]	SELECTABLE attribute
SELECTED attribute	SELECTION-END attribute	SELECTION-START attribute
SELECTION-TEXT attribute	SENSITIVE attribute	SIDE-LABEL-HANDLE attribute
SUBTYPE attribute	TABLE attribute	TAB-POSITION attribute
TAB-STOP attribute	TEXT-SELECTED attribute	TOOLTIP attribute
TYPE attribute[71]	VISIBLE attribute	WIDGET-ID attribute
WIDTH-CHARS attribute[71]	WIDTH-PIXELS attribute[71]	WINDOW attribute
X attribute[71]	Y attribute[71]	–

Methods

CLEAR( ) method (FILL-IN Widget)	CLEAR-SELECTION( ) method[72]

[72] This method also applies to fill-in browse columns.

EDIT-CLEAR( ) method[72]	EDIT-COPY( ) method[72]
EDIT-CUT( ) method[72]	EDIT-PASTE( ) method[72]
EDIT-UNDO( ) method[72]	END-FILE-DROP( ) method
GET-DROPPED-FILE( ) method	LOAD-MOUSE-POINTER( ) method
MOVE-AFTER-TAB-ITEM( ) method	MOVE-BEFORE-TAB-ITEM( ) method
MOVE-TO-BOTTOM( ) method	MOVE-TO-TOP( ) method
SET-SELECTION( ) method[72]	VALIDATE( ) method

Events

Default keyboard events	Developer events
Field editing key function events	General direct manipulation events
Mouse events	Navigation key function events
Universal key function events	DROP-FILE-NOTIFY
ENTRY	LEAVE
VALUE-CHANGED	–

See also

CREATE widget statement, VIEW-AS phrase

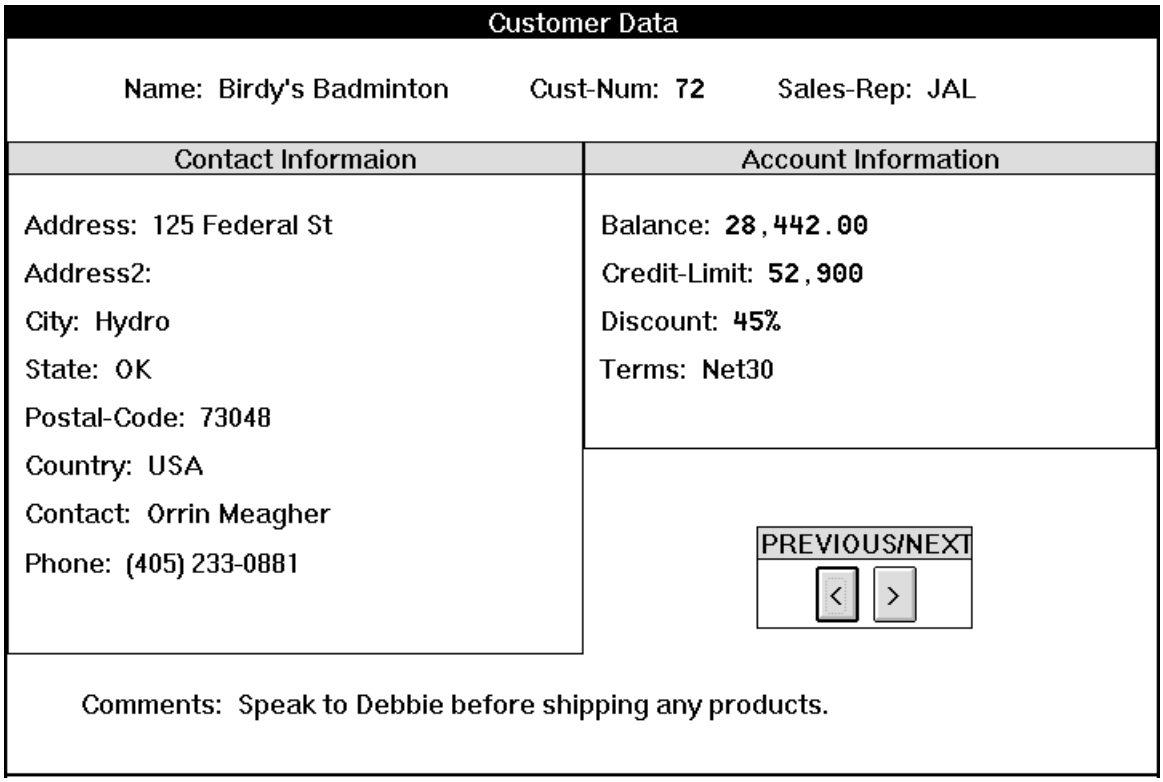
FRAME widget

A frame is a display area within a window that can group together (contain) a set of field-level widgets and child frames. In addition to default frames set up by ABL, you can set up static frames with the Frame phrase or DEFINE FRAME statement. You can create a dynamic one-down frame with the CREATE widget statement.

Related field-level widgets and child frames are actually parented by a single field group widget, which is owned, in turn, by the parenting frame. You parent static field-level widgets to a static frame using a DEFINE FRAME, FORM, or FRAME I/O statement. You parent dynamic field-level widgets to any frame by setting the FRAME attribute of each field-level widget to the handle of the parent frame. You can parent frame widgets to any frame by setting the FRAME attribute of each child frame to the handle of its parent frame.

Frames in a parent and child relationship form a frame family, which is a hierarchy of parent and child frames ultimately parented by a window. The top parent frame that is parented by the window is the root frame of the frame family.

The following figure shows a frame family with four frames, including three child frames titled Contact Information, Account Information, and PREVIOUS/NEXT:



Attributes

BACKGROUND attribute	BGCOLOR attribute	BLOCK-ITERATION-DISPLAY attribute
BORDER-BOTTOM-CHARS attribute	BORDER-BOTTOM-PIXELS attribute	BORDER-LEFT-CHARS attribute
BORDER-LEFT-PIXELS attribute	BORDER-RIGHT-CHARS attribute	BORDER-RIGHT-PIXELS attribute
BORDER-TOP-CHARS attribute	BORDER-TOP-PIXELS attribute	BOX attribute
BOX-SELECTABLE attribute	CANCEL-BUTTON attribute	CAREFUL-PAINT attribute
CENTERED attribute	COLUMN attribute	CURRENT-ITERATION attribute (Widget Objects)
DCOLOR attribute	DDE-ERROR attribute	DDE-ID attribute
DDE-ITEM attribute	DDE-NAME attribute	DDE-TOPIC attribute
DEFAULT-BUTTON attribute	DOWN attribute	DROP-TARGET attribute
DYNAMIC attribute	FGCOLOR attribute	FIRST-CHILD attribute
FONT attribute	FRAME attribute	GRID-FACTOR-HORIZONTAL attribute

GRID-FACTOR-VERTICAL attribute	GRID-SNAP attribute	GRID-UNIT-HEIGHT-CHARS attribute
GRID-UNIT-HEIGHT-PIXELS attribute	GRID-UNIT-WIDTH-CHARS attribute	GRID-UNIT-WIDTH-PIXELS attribute
GRID-VISIBLE attribute	HANDLE attribute	HEIGHT-CHARS attribute
HEIGHT-PIXELS attribute	HIDDEN attribute	HTML-CHARSET attribute
HWND attribute	INHERIT-BGCOLOR attribute	INHERIT-FGCOLOR attribute
INSTANTIATING-PROCEDURE attribute	LABELS attribute	LAST-CHILD attribute
LINE attribute	MANUAL-HIGHLIGHT attribute	MENU-KEY attribute
MENU-MOUSE attribute	MOUSE-POINTER attribute	MOVABLE attribute
NAME attribute	NEXT-SIBLING attribute	NEXT-TAB-ITEM attribute
NUM-DROPPED-FILES attribute	NUM-SELECTED-WIDGETS attribute	NUM-TO-RETAIN attribute
OVERLAY attribute	PAGE-BOTTOM attribute	PAGE-TOP attribute
PARENT attribute	PFCOLOR attribute	PIXELS-PER-COLUMN attribute
PIXELS-PER-ROW attribute	POPUP-MENU attribute	PREV-SIBLING attribute
PREV-TAB-ITEM attribute	PRIVATE-DATA attribute	RESIZABLE attribute
ROW attribute	SCROLLABLE attribute	SELECTABLE attribute
SELECTED attribute	SENSITIVE attribute	SIDE-LABELS attribute
TAB-POSITION attribute	TAB-STOP attribute	THREE-D attribute
TITLE attribute	TITLE-BGCOLOR attribute	TITLE-DCOLOR attribute
TITLE-FGCOLOR attribute	TITLE-FONT attribute	TOP-ONLY attribute
TYPE attribute	VIRTUAL-HEIGHT-CHARS attribute	VIRTUAL-HEIGHT-PIXELS attribute
VIRTUAL-WIDTH-CHARS attribute	VIRTUAL-WIDTH-PIXELS attribute	VISIBLE attribute
WIDGET-ID attribute	WIDTH-CHARS attribute	WIDTH-PIXELS attribute
WINDOW attribute	X attribute	Y attribute

Methods

END-FILE-DROP( ) method	GET-DROPPED-FILE( ) method

GET-ITERATION( ) method (Widget Objects)	GET-SELECTED-WIDGET( ) method
LOAD-MOUSE-POINTER( ) method	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	VALIDATE( ) method

Events

Developer events	Frame-only direct manipulation events
General direct manipulation events	Mouse events
Universal key function events	DDE-NOTIFY[73]
DROP-FILE-NOTIFY	ENTRY
LEAVE	–

Notes

- Field-level widgets and child frames are not directly parented by a parent frame. They are parented by field groups that are owned by the parent frame. Thus, you can also parent a child frame by setting the child frame's PARENT attribute to the handle of a field group in the parent frame.

  To access all the field-level widgets and child frames owned by a frame, you must first use the frame's FIRST-CHILD or LAST-CHILD attribute to find a field group within the frame. You can then use the field group's NEXT-SIBLING or PREV-SIBLING attribute to find other field groups in the frame. You can use the field group's FIRST-CHILD or LAST-CHILD attribute to find a field-level widget or child frame within the field group. You can then use the field-level widget's or child frame's NEXT-SIBLING or PREV-SIBLING attribute to find other field-level widgets and child frames within the frame.

- Child frames do not inherit the attributes of a parent frame.
- When any of a frame's field-level widgets or child frames are viewed using the DISPLAY or ENABLE statement, the parent frame also becomes visible unless its HIDDEN attribute or the HIDDEN attribute of an ancestor widget is TRUE. However, explicitly setting the VISIBLE attribute to TRUE (using the VIEW statement) for a child frame or field-level widget makes all ancestor frames visible, unless the parent or an ancestor window has its HIDDEN attribute set to TRUE.
- Child frames participate in the tab order along with any field-level widgets in the same parent frame. This means that the tab orders of all field-level widgets within a child frame is placed as a group within the tab order of the siblings of that child frame. Thus, tabbing proceeds between the field-level widgets of a root frame and the field-level widgets of all descendant frames. However, tabbing is not supported between sibling root frames (frames parented by a window).
- You specify the position of a child frame relative to the display area of the parent frame. You must specify the position so that the upper left corner of the child frame lies within the display region of the parent frame. Otherwise at run time, when the procedure tries to realize the frame, the AVM raises the ERROR condition.
- When you apply a NEXT-FRAME or PREV-FRAME navigation key function to a field-level widget, focus changes from the current frame family to the next or previous frame family (respectively) parented by the

[73] Windows only. This event occurs only in dynamic data exchange (DDE) conversations. This event is supported only for backward compatibility. Use the Component Object Model (COM) instead. For more information, see the topics on DDE in *OpenEdge Programming Interfaces*

same window. That is, these key functions change focus between root frames, not between descendant frames.

- In character interfaces, the SCROLL-MODE function key is available for a frame only if the SCROLLABLE attribute of the frame is TRUE. Scroll mode allows you to use the CURSOR-RIGHT and CURSOR-LEFT keys to scroll the frame horizontally. The SCROLL-MODE function key toggles scroll mode on and off for a frame that has focus.

See also

CREATE widget statement, DIALOG-BOX widget, DEFINE FRAME statement, Frame phrase

---

IMAGE widget
(Graphical interfaces only)

An image is a graphic taken from an operating system file. It can be used by itself or within a button. You can define a static image with the DEFINE IMAGE statement, and create a dynamic image with the CREATE widget statement. You can specify an image for a button using the DEFINE BUTTON statement or the button methods for loading images.



Attributes

BGCOLOR attribute	COLUMN attribute	CONVERT-3D-COLORS attribute
DYNAMIC attribute	FGCOLOR attribute	FRAME attribute
FRAME-COL attribute	FRAME-NAME attribute	FRAME-ROW attribute
FRAME-X attribute	FRAME-Y attribute	HANDLE attribute
HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute	HELP attribute
HIDDEN attribute	HTML-CHARSET attribute	IMAGE attribute
INSTANTIATING-PROCEDURE attribute	MANUAL-HIGHLIGHT attribute	MOVABLE attribute
NAME attribute	NEXT-SIBLING attribute	PARENT attribute
PREV-SIBLING attribute	PRIVATE-DATA attribute	RESIZABLE attribute
RETAIN-SHAPE attribute	ROW attribute	SELECTABLE attribute
SELECTED attribute	SENSITIVE attribute	STRETCH-TO-FIT attribute
TOOLTIP attribute	TRANSPARENT attribute	TYPE attribute
VISIBLE attribute	WIDGET-ID attribute	WIDTH-CHARS attribute

WIDTH-PIXELS attribute	WINDOW attribute	X attribute
Y attribute	–	–

Methods

LOAD-IMAGE( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	–

Events

Developer events	General direct manipulation events
Mouse events	–

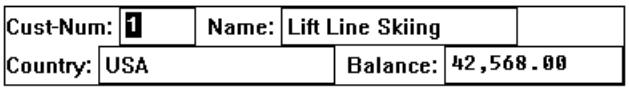
See also

CREATE widget statement, DEFINE BUTTON statement, DEFINE IMAGE statement

---

LITERAL widget

A literal widget is the label for a static field. If a field has a side label, you can find the handle of a literal widget by reading the field's SIDE-LABEL-HANDLE attribute. If the field has a column label, you can find the handle of the literal by examining the children of the frame's background field group. You cannot create a literal widget dynamically.



Attributes

BGCOLOR attribute	COLUMN attribute	DCOLOR attribute
DYNAMIC attribute	FGCOLOR attribute	FONT attribute
FRAME attribute	FRAME-COL attribute	FRAME-NAME attribute
FRAME-ROW attribute	FRAME-X attribute	FRAME-Y attribute
HANDLE attribute	HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute
HIDDEN attribute	HTML-CHARSET attribute	INPUT-VALUE attribute
INSTANTIATING-PROCEDURE attribute	MANUAL-HIGHLIGHT attribute	MOVABLE attribute
NAME attribute	NEXT-SIBLING attribute	PARENT attribute

PREV-SIBLING attribute	PRIVATE-DATA attribute	RESIZABLE attribute
ROW attribute	SCREEN-VALUE attribute	SELECTABLE attribute
SELECTED attribute	SENSITIVE attribute	TYPE attribute
VISIBLE attribute	WIDTH-CHARS attribute	WIDTH-PIXELS attribute
WINDOW attribute	X attribute	Y attribute

Methods

MOVE-TO-BOTTOM( ) method	MOVE-TO-TOP( ) method

Events

The literal widget does not support any events.

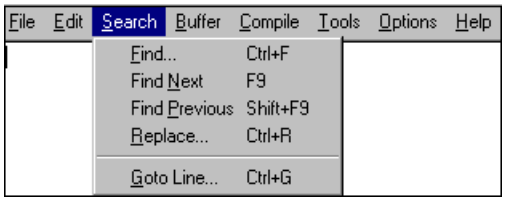
See also

SIDE-LABEL-HANDLE attribute

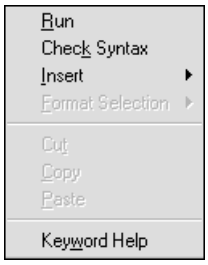
MENU widget

A menu can be a menu bar or a pop-up menu. Menu bars contain sub-menus (specifically, pull-down menus) and in some environments menu items. Pop-up menus contain menu items and sub-menus. You can define a static menu with the DEFINE MENU statement. You can create a dynamic menu with the CREATE widget statement.

The following is a menu bar:



The following is a pop-up menu:



Attributes

DCOLOR attribute	DYNAMIC attribute	FIRST-CHILD attribute

HANDLE attribute	HTML-CHARSET attribute	INSTANTIATING-PROCEDURE attribute
LAST-CHILD attribute	NAME attribute	OWNER attribute
PFCOLOR attribute	POPUP-ONLY attribute	PRIVATE-DATA attribute
SENSITIVE attribute	TITLE attribute	TITLE-DCOLOR attribute
TYPE attribute	VISIBLE attribute	WINDOW attribute

Note: Color and font attributes for a menu are ignored in Windows.

Methods

The MENU widget does not support any methods.

Events

Developer events	MENU-DROP [74]

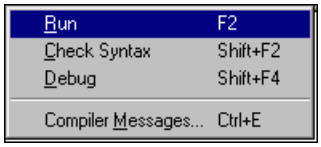
See also

CREATE widget statement, DEFINE MENU statement

MENU-ITEM widget

A menu item is an item within a menu or submenu. A menu item can be a rule, a space, or a normal menu item. A normal menu item can be a command or a toggle-box item. Most menu item attributes and all menu item events apply only to normal menu items. You can set up a static menu item within a DEFINE MENU statement or DEFINE SUB-MENU statement. You can create a dynamic menu item with the CREATE widget statement.

The following is a menu containing four menu items:



Attributes

ACCELERATOR attribute	CHECKED attribute	DCOLOR attribute
DYNAMIC attribute	HANDLE attribute	HTML-CHARSET attribute
INSTANTIATING-PROCEDURE attribute	LABEL attribute	NAME attribute

[74] Supported only when the POPUP-ONLY attribute is set to TRUE and the menu is set as a popup for some other widget.

NEXT-SIBLING attribute	PARENT attribute	PFCOLOR attribute
PREV-SIBLING attribute	PRIVATE-DATA attribute	READ-ONLY attribute
SENSITIVE attribute	SUBTYPE attribute	TOGGLE-BOX attribute
TYPE attribute	VISIBLE attribute	WINDOW attribute

Note: Color and font attributes for a menu item are ignored in Windows.

Methods

The MENU-ITEM widget does not support any methods.

Events

Developer events	CHOOSE (except for toggle-box items)
VALUE-CHANGED (for toggle-box items only)	–

See also

CREATE widget statement, DEFINE MENU statement, DEFINE SUB-MENU statement

RADIO-SET widget

A radio set is a group of values of which only one can be set at any time. You can define a static radio set by using the VIEW-AS phrase with any LOGICAL, CHARACTER, INTEGER, INT64, DECIMAL, or DATE value. You can create a dynamic radio set with the CREATE widget statement. For example:



Attributes

AUTO-RESIZE attribute	BGCOLOR attribute	COLUMN attribute
CONTEXT-HELP-ID attribute	DATA-TYPE attribute	DBNAME attribute
DCOLOR attribute	DELIMITER attribute	DROP-TARGET attribute
DYNAMIC attribute	EXPAND attribute	FGCOLOR attribute
FONT attribute	FRAME attribute	FRAME-COL attribute
FRAME-NAME attribute	FRAME-ROW attribute	FRAME-X attribute

FRAME-Y attribute	HANDLE attribute	HEIGHT-CHARS attribute
HEIGHT-PIXELS attribute	HELP attribute	HIDDEN attribute
HORIZONTAL attribute	HTML-CHARSET attribute	INPUT-VALUE attribute
INSTANTIATING-PROCEDURE attribute	LABEL attribute	MANUAL-HIGHLIGHT attribute
MENU-KEY attribute	MENU-MOUSE attribute	MODIFIED attribute
MOUSE-POINTER attribute	MOVABLE attribute	NAME attribute
NEXT-SIBLING attribute	NEXT-TAB-ITEM attribute	NUM-BUTTONS attribute
NUM-DROPPED-FILES attribute	PARENT attribute	PFCOLOR attribute
POPUP-MENU attribute	PREV-SIBLING attribute	PREV-TAB-ITEM attribute
PRIVATE-DATA attribute	RADIO-BUTTONS attribute	RESIZABLE attribute
ROW attribute	SCREEN-VALUE attribute	SELECTABLE attribute
SELECTED attribute	SENSITIVE attribute	SIDE-LABEL-HANDLE attribute
TABLE attribute	TAB-POSITION attribute	TAB-STOP attribute
TOOLTIP attribute	TYPE attribute	VISIBLE attribute
WIDGET-ID attribute	WIDTH-CHARS attribute	WIDTH-PIXELS attribute
WINDOW attribute	X attribute	Y attribute

Methods

ADD-LAST( ) method	DELETE( ) method
DISABLE( ) method	ENABLE( ) method
END-FILE-DROP( ) method	GET-DROPPED-FILE( ) method
LOAD-MOUSE-POINTER( ) method	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	REPLACE( ) method
VALIDATE( ) method	–

Events

Default keyboard events	Developer events

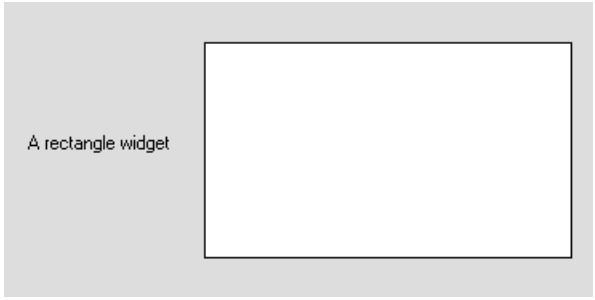
General direct manipulation events	Mouse events
Navigation key function events	Universal key function events
DROP-FILE-NOTIFY	ENTRY
LEAVE	–

See also

CREATE widget statement, VIEW-AS phrase

RECTANGLE widget

A rectangle is a graphical widget that can be displayed in a frame foreground or background. You can define a static rectangle with the DEFINE RECTANGLE statement. You can create a dynamic rectangle with the CREATE widget statement. For example:





Attributes

BGCOLOR attribute	COLUMN attribute	DATA-TYPE attribute
DCOLOR attribute	DYNAMIC attribute	EDGE-CHARS attribute
EDGE-PIXELS attribute	FGCOLOR attribute	FILLED attribute
FRAME attribute	FRAME-COL attribute	FRAME-NAME attribute
FRAME-ROW attribute	FRAME-X attribute	FRAME-Y attribute
GRAPHIC-EDGE attribute	GROUP-BOX attribute	HANDLE attribute
HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute	HELP attribute

HIDDEN attribute	HTML-CHARSET attribute	INSTANTIATING-PROCEDURE attribute
MANUAL-HIGHLIGHT attribute	MOVABLE attribute	NAME attribute
NEXT-SIBLING attribute	PARENT attribute	PFCOLOR attribute
PREV-SIBLING attribute	PRIVATE-DATA attribute	RESIZABLE attribute
ROUNDED attribute	ROW attribute	SELECTABLE attribute
SELECTED attribute	SENSITIVE attribute	TABLE attribute
TOOLTIP attribute	TYPE attribute	VISIBLE attribute
WIDGET-ID attribute	WIDTH-CHARS attribute	WIDTH-PIXELS attribute
WINDOW attribute	X attribute	Y attribute

Methods

MOVE-TO-BOTTOM( ) method	MOVE-TO-TOP( ) method

Events

Developer events	General direct manipulation events
Mouse events	–

See also

CREATE widget statement, DEFINE RECTANGLE statement

SELECTION-LIST widget

A selection list is a widget that contains a list of possible values for a field or variable. You can use the VIEW-AS phrase to set up a static selection list. You can use the CREATE widget statement to create a dynamic selection list. For example:



Attributes

AUTO-RESIZE attribute	BGCOLOR attribute	COLUMN attribute
CONTEXT-HELP-ID attribute	DATA-TYPE attribute	DBNAME attribute
DCOLOR attribute	DELIMITER attribute	DRAG-ENABLED attribute
DROP-TARGET attribute	DYNAMIC attribute	FGCOLOR attribute
FONT attribute	FRAME attribute	FRAME-COL attribute
FRAME-NAME attribute	FRAME-ROW attribute	FRAME-X attribute
FRAME-Y attribute	HANDLE attribute	HEIGHT-CHARS attribute
HEIGHT-PIXELS attribute	HELP attribute	HIDDEN attribute
HTML-CHARSET attribute	INNER-CHARS attribute	INNER-LINES attribute
INPUT-VALUE attribute	INSTANTIATING-PROCEDURE attribute	LABEL attribute
LIST-ITEM-PAIRS attribute	LIST-ITEMS attribute	MANUAL-HIGHLIGHT attribute
MENU-KEY attribute	MENU-MOUSE attribute	MODIFIED attribute
MOUSE-POINTER attribute	MOVABLE attribute	MULTIPLE attribute
NAME attribute	NEXT-SIBLING attribute	NEXT-TAB-ITEM attribute
NUM-DROPPED-FILES attribute	NUM-ITEMS attribute	PARENT attribute
PFCOLOR attribute	POPUP-MENU attribute	PREV-SIBLING attribute
PREV-TAB-ITEM attribute	PRIVATE-DATA attribute	RESIZABLE attribute
ROW attribute	SCREEN-VALUE attribute	SCROLLBAR-HORIZONTAL attribute
SCROLLBAR-VERTICAL attribute	SELECTABLE attribute	SELECTED attribute
SENSITIVE attribute	SIDE-LABEL-HANDLE attribute	SORT attribute
TABLE attribute	TAB-POSITION attribute	TAB-STOP attribute
TOOLTIP attribute	TYPE attribute	VISIBLE attribute
WIDGET-ID attribute	WIDTH-CHARS attribute	WIDTH-PIXELS attribute
WINDOW attribute	X attribute	Y attribute

Methods

ADD-FIRST( ) method	ADD-LAST( ) method
DELETE( ) method)	END-FILE-DROP( ) method
ENTRY( ) method	GET-DROPPED-FILE( ) method
INSERT( ) method	IS-SELECTED( ) method
LOAD-MOUSE-POINTER( ) method	LOOKUP( ) method
MOVE-AFTER-TAB-ITEM( ) method	MOVE-BEFORE-TAB-ITEM( ) method
MOVE-TO-BOTTOM( ) method	MOVE-TO-TOP( ) method
REPLACE( ) method	SCROLL-TO-ITEM( ) method
VALIDATE( ) method	–

Events

Default keyboard events	Developer events
General direct manipulation events	Mouse events
Navigation key function events	Universal key function events
DEFAULT-ACTION	DROP-FILE-NOTIFY
ENTRY	LEAVE
VALUE-CHANGED	–

See also

CREATE widget statement, VIEW-AS phrase

SHADOW-WINDOW widget
(Windows only; GUI for .NET only)

A shadow window widget is a window associated with a .NET form that allows .NET forms and ABL windows to have parenting relationships to each other. A shadow window has no visible representation. It exists solely to support a .NET form and its relationship to ABL windows in an ABL session.

You cannot explicitly create or delete a shadow window. The ABL virtual machine (AVM) automatically creates an associated shadow window when you instantiate a .NET form from the OpenEdge .NET Progress.Windows.Form class, and it automatically deletes the associated shadow window when you delete an instance `Progress.Windows.Form`.

Attributes

PARENT attribute	–

Methods

The SHADOW-WINDOW widget does not support any methods.

Events

The SHADOW-WINDOW widget does not support any events.

Notes

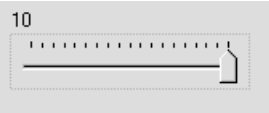
- To obtain the shadow window handle of a .NET form, read the `ProWinHandle` property of the associated `Progress.Windows.Form` instance.
- To parent an ABL window to a .NET form, assign the shadow window handle of the .NET form to the PARENT attribute of the ABL window.
- To parent a .NET form to an ABL window, assign the handle of the ABL window to the PARENT attribute of the shadow window.
- Shadow windows have no NEXT-SIBLING attribute or PREV-SIBLING attribute and therefore do not appear on the session window chain.
- ABL does not create shadow windows for .NET forms that you instantiate from the `System.Windows.Forms.Form` class. Progress Software Corporation recommends that you always instantiate such forms from `Progress.Windows.Form`.

See also

Progress.Windows.Form class, ProWinHandle property, WINDOW widget

SLIDER widget

The slider widget represents an integer value as a point on a sliding scale. You can use the VIEW-AS phrase to set up a static slider. You can use the CREATE widget statement to create a dynamic slider. For example:



Attributes

AUTO-RESIZE attribute	BGCOLOR attribute	COLUMN attribute
CONTEXT-HELP-ID attribute	DATA-TYPE attribute	DBNAME attribute
DCOLOR attribute	DROP-TARGET attribute	DYNAMIC attribute
FGCOLOR attribute	FONT attribute	FRAME attribute
FRAME-COL attribute	FRAME-NAME attribute	FRAME-ROW attribute

FRAME-X attribute	FRAME-Y attribute	FREQUENCY attribute
HANDLE attribute	HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute
HELP attribute	HIDDEN attribute	HORIZONTAL attribute
HTML-CHARSET attribute	INPUT-VALUE attribute	INSTANTIATING-PROCEDURE attribute
LABEL attribute	LARGE-TO-SMALL attribute	MANUAL-HIGHLIGHT attribute
MAX-VALUE attribute	MENU-KEY attribute	MENU-MOUSE attribute
MIN-VALUE attribute	MODIFIED attribute	MOUSE-POINTER attribute
MOVABLE attribute	NAME attribute	NEXT-SIBLING attribute
NEXT-TAB-ITEM attribute	NO-CURRENT-VALUE attribute	NUM-DROPPED-FILES attribute
PARENT attribute	PFCOLOR attribute	POPUP-MENU attribute
PREV-SIBLING attribute	PREV-TAB-ITEM attribute	PRIVATE-DATA attribute
RESIZABLE attribute	ROW attribute	SCREEN-VALUE attribute
SELECTABLE attribute	SELECTED attribute	SENSITIVE attribute
SIDE-LABEL-HANDLE attribute	TABLE attribute	TAB-POSITION attribute
TAB-STOP attribute	TIC-MARKS attribute	TOOLTIP attribute
TYPE attribute	VISIBLE attribute	WIDGET-ID attribute
WIDTH-CHARS attribute	WIDTH-PIXELS attribute	WINDOW attribute
X attribute	Y attribute	–

Methods

END-FILE-DROP( ) method	GET-DROPPED-FILE( ) method
LOAD-MOUSE-POINTER( ) method	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	VALIDATE( ) method

Events

Default keyboard events	Developer events
General direct manipulation events	Mouse events

Navigation key function events	Universal key function events
DROP-FILE-NOTIFY	ENTRY
LEAVE	VALUE-CHANGED

Notes

- Only a value of the INTEGER or INT64 data type can be viewed as a slider. If using INT64, the value must remain within the -2147483648 to +2147483647 range.
- In character interfaces, a slider widget has a minimum width that is dependent on the specified maximum value (MAX-VALUE attribute). The minimum height for a slider widget in a character interface is 2 character units. You can specify a value as low as 1.5 character units for the height of a slider in a character interface; however, ABL rounds the value up to 2 character units.

See also

CREATE widget statement, VIEW-AS phrase

SUB-MENU widget

A submenu can be a pull-down menu within a menu bar, or a submenu within a pull-down menu or pop-up menu. You can define a static submenu with the DEFINE SUB-MENU statement. You can use the CREATE widget statement to create a dynamic submenu. For example:



Attributes

BGCOLOR attribute	DCOLOR attribute	DYNAMIC attribute
FGCOLOR attribute	FIRST-CHILD attribute	FONT attribute
HANDLE attribute	HTML-CHARSET attribute	INSTANTIATING-PROCEDURE attribute
LABEL attribute	LAST-CHILD attribute	NAME attribute
NEXT-SIBLING attribute	PARENT attribute	PFCOLOR attribute
PREV-SIBLING attribute	PRIVATE-DATA attribute	SENSITIVE attribute
TYPE attribute	VISIBLE attribute	WINDOW attribute

Note: Color and font attributes for a submenu are ignored in Windows.

Methods

The SUB-MENU widget does not support any methods.

Events

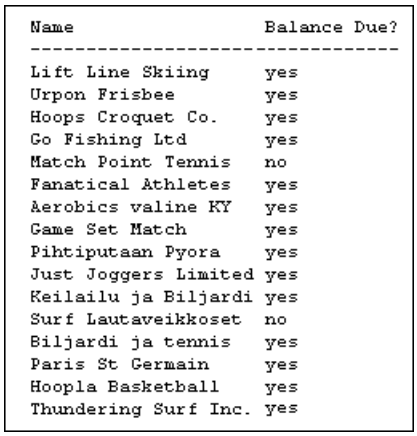
Developer events	MENU-DROP

See also

CREATE widget statement, DEFINE SUB-MENU statement

TEXT widget

You can use the text widget to display read-only text in a compact format. This is especially useful when you are creating hard-copy reports. You can use the VIEW-AS phrase to set up a static text widget. You can use the CREATE widget statement to create dynamic text widgets. For example:



```
Name                 Balance Due?
---------------------------------
Lift Line Skiing     yes
Urpon Frisbee        yes
Hoops Croquet Co.    yes
Go Fishing Ltd       yes
Match Point Tennis   no
Fanatical Athletes   yes
Aerobics valine KY   yes
Game Set Match       yes
Pihtiputaan Pyora    yes
Just Joggers Limited yes
Keilailu ja Biljardi yes
Surf Lautaveikkoset  no
Biljardi ja tennis   yes
Paris St Germain     yes
Hoopla Basketball    yes
Thundering Surf Inc. yes
```

Attributes

ATTR-SPACE attribute	AUTO-RESIZE attribute	BGCOLOR attribute
BLANK attribute	COLUMN attribute	DATA-TYPE attribute
DBNAME attribute	DCOLOR attribute	DEBLANK attribute
DYNAMIC attribute	FGCOLOR attribute	FONT attribute
FORMAT attribute	FRAME attribute	FRAME-COL attribute
FRAME-NAME attribute	FRAME-ROW attribute	FRAME-X attribute
FRAME-Y attribute	HANDLE attribute	HEIGHT-CHARS attribute
HEIGHT-PIXELS attribute	HELP attribute	HIDDEN attribute
HTML-CHARSET attribute	INDEX attribute	INPUT-VALUE attribute

INSTANTIATING-PROCEDURE attribute	LABEL attribute	LABELS attribute
MANUAL-HIGHLIGHT attribute	MODIFIED attribute	MOVABLE attribute
NAME attribute	NEXT-SIBLING attribute	PARENT attribute
PREV-SIBLING attribute	PRIVATE-DATA attribute	RESIZABLE attribute
ROW attribute	SCREEN-VALUE attribute	SELECTABLE attribute
SELECTED attribute	SENSITIVE attribute	SIDE-LABEL-HANDLE attribute
TABLE attribute	TOOLTIP attribute	TYPE attribute
VISIBLE attribute	WIDGET-ID attribute	WIDTH-CHARS attribute
WIDTH-PIXELS attribute	WINDOW attribute	X attribute
Y attribute	–	–

Methods

MOVE-TO-BOTTOM( ) method	MOVE-TO-TOP( ) method

Events

Developer events	General direct manipulation events
Mouse events	–

Note

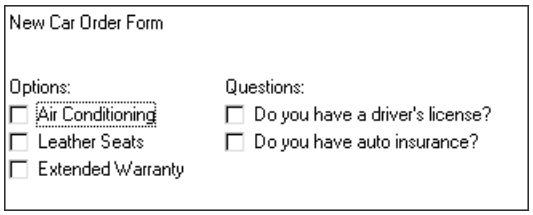
You can view a field as text by specifying VIEW-AS TEXT for the field. You can make text the default representation for all fields in a frame by specifying USE-TEXT for the frame.

See also

CREATE widget statement, VIEW-AS phrase

TOGGLE-BOX widget

You can use the toggle box widget to represent a logical value. You can use the VIEW-AS phrase to set up a static toggle box, or the CREATE widget statement to create a dynamic toggle box. This figure shows five toggle boxes:



Attributes

AUTO-RESIZE attribute	BGCOLOR attribute[75]	CHECKED attribute
COLUMN attribute	CONTEXT-HELP-ID attribute	DATA-TYPE attribute
DBNAME attribute	DCOLOR attribute[75]	DROP-TARGET attribute
DYNAMIC attribute	FGCOLOR attribute[75]	FONT attribute
FORMAT attribute	FRAME attribute	FRAME-COL attribute
FRAME-NAME attribute	FRAME-ROW attribute	FRAME-X attribute
FRAME-Y attribute	HANDLE attribute[75]	HEIGHT-CHARS attribute[75]
HEIGHT-PIXELS attribute[75]	HELP attribute	HIDDEN attribute
HWND attribute	INDEX attribute	INPUT-VALUE attribute[75]
INSTANTIATING-PROCEDURE attribute	LABEL attribute	MANUAL-HIGHLIGHT attribute
MENU-KEY attribute	MENU-MOUSE attribute	MODIFIED attribute
MOUSE-POINTER attribute	MOVABLE attribute	NAME attribute[75]
NEXT-SIBLING attribute	NEXT-TAB-ITEM attribute	NUM-DROPPED-FILES attribute
PARENT attribute	PFCOLOR attribute[75]	POPUP-MENU attribute
PREV-SIBLING attribute	PREV-TAB-ITEM attribute	PRIVATE-DATA attribute
RESIZABLE attribute	ROW attribute[75]	SCREEN-VALUE attribute[75]
SELECTABLE attribute	SELECTED attribute	SENSITIVE attribute
TABLE attribute	TAB-POSITION attribute	TAB-STOP attribute
TOOLTIP attribute	TYPE attribute[75]	VISIBLE attribute
WIDGET-ID attribute	WIDTH-CHARS attribute[75]	WIDTH-PIXELS attribute[75]
WINDOW attribute	X attribute[75]	Y attribute[75]

[75] This attribute also applies to toggle-box browse columns.

Methods

END-FILE-DROP( ) method	GET-DROPPED-FILE( ) method
LOAD-MOUSE-POINTER( ) method	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	VALIDATE( ) method

Events

Default keyboard events	Developer events
General direct manipulation events	Mouse events
Navigation key function events	Universal key function events
DROP-FILE-NOTIFY	ENTRY
LEAVE	VALUE-CHANGED

See also

CREATE widget statement, VIEW-AS phrase

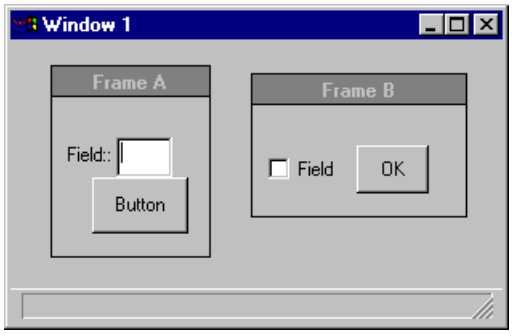
WINDOW widget

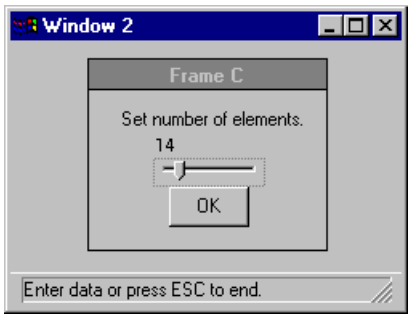
A window is a rectangular area on the screen that can contain frame widgets, parent dialog boxes, and parent other windows. It is surrounded by a standard border and affordances provided by your window system to manipulate the window's size, location, and appearance on the screen.

The AVM automatically creates one default window for each session. You can create additional dynamic windows with the CREATE widget statement. Each additional window can be parented by the window system, creating siblings (the default) or by another window, creating child and parent window relationships. You create a parent and child relationship between two windows by setting the PARENT attribute of one (the child) to the handle of the other (the parent).

Windows in a parent and child relationship form a window family, which is a hierarchy of parent and child windows ultimately parented by the window system. The top parent window that is parented by the window system is the root window of the window family.

The following figure shows a window family consisting of a root window and its child window:





Attributes

ALWAYS-ON-TOP attribute	BGCOLOR attribute	COLUMN attribute
CONTEXT-HELP attribute	CONTEXT-HELP-FILE attribute	CONTROL-BOX attribute
DCOLOR attribute	DROP-TARGET attribute	DYNAMIC attribute
FGCOLOR attribute	FIRST-CHILD attribute	FONT attribute
FULL-HEIGHT-CHARS attribute	FULL-HEIGHT-PIXELS attribute	FULL-WIDTH-CHARS attribute
FULL-WIDTH-PIXELS attribute	HANDLE attribute	HEIGHT-CHARS attribute
HEIGHT-PIXELS attribute	HIDDEN attribute	HWND attribute
ICON attribute	INSTANTIATING-PROCEDURE attribute	KEEP-FRAME-Z-ORDER attribute
LAST-CHILD attribute	MAX-BUTTON attribute	MAX-HEIGHT-CHARS attribute
MAX-HEIGHT-PIXELS attribute	MAX-WIDTH-CHARS attribute	MAX-WIDTH-PIXELS attribute
MENU-BAR attribute	MENU-KEY attribute	MENU-MOUSE attribute
MESSAGE-AREA attribute	MESSAGE-AREA-FONT attribute	MIN-BUTTON attribute
MIN-HEIGHT-CHARS attribute	MIN-HEIGHT-PIXELS attribute	MIN-WIDTH-CHARS attribute
MIN-WIDTH-PIXELS attribute	MOUSE-POINTER attribute	NAME attribute
NEXT-SIBLING attribute	NUM-DROPPED-FILES attribute	NUM-SELECTED-WIDGETS attribute
PARENT attribute	PFCOLOR attribute	POPUP-MENU attribute
PREV-SIBLING attribute	PRIVATE-DATA attribute	RESIZE attribute
ROW attribute	SCREEN-LINES attribute	SCROLL-BARS attribute
SENSITIVE attribute	SHOW-IN-TASKBAR attribute	SMALL-ICON attribute
SMALL-TITLE attribute	STATUS-AREA attribute	STATUS-AREA-FONT attribute

THREE-D attribute	TITLE attribute	TOP-ONLY attribute
TYPE attribute	VIRTUAL-HEIGHT-CHARS attribute	VIRTUAL-HEIGHT-PIXELS attribute
VIRTUAL-WIDTH-CHARS attribute	VIRTUAL-WIDTH-PIXELS attribute	VISIBLE attribute
WIDTH-CHARS attribute	WIDTH-PIXELS attribute	WINDOW attribute
WINDOW-STATE attribute	X attribute	Y attribute

Methods

END-FILE-DROP( ) method	GET-DROPPED-FILE( ) method
GET-SELECTED-WIDGET( ) method	LOAD-ICON( ) method
LOAD-MOUSE-POINTER( ) method	LOAD-SMALL-ICON( ) method
MOVE-TO-BOTTOM( ) method	MOVE-TO-TOP( ) method

Events

Developer events	Mouse events
Universal key function events	DROP-FILE-NOTIFY
ENTRY	LEAVE
PARENT-WINDOW-CLOSE	WINDOW-CLOSE
WINDOW-MAXIMIZED	WINDOW-MINIMIZED
WINDOW-RESIZED	WINDOW-RESTORED

Notes

- In character user interfaces, you can have only one window (the default window).
- Certain manipulations of a parent window have a default effect on its child windows and their descendants. You cannot modify the following effects:
  - Iconifying a window (triggered by a WINDOW-MINIMIZED event or by setting the WINDOW-STATE attribute to WINDOW-MINIMIZED) causes any of its descendant windows that are not already iconified to be hidden. Any child windows that are already iconified remain iconified along with their parents. Restoring the parent window (triggered by a WINDOW-RESTORED event) causes all of its descendant windows to receive a WINDOW-RESTORED event, restoring them to their visual state prior to the parent window being minimized.
  - When a window receives a WINDOW-MINIMIZED event, all of its descendant windows receive WINDOW-MINIMIZED events. When a window receives a WINDOW-RESTORED event, all of its descendant windows receive WINDOW-RESTORED events.
  - Applying a WINDOW-CLOSE event to a window causes all of its descendant windows to receive a PARENT-WINDOW-CLOSE event. However, any ancestor windows remain unaffected.

- A WINDOW-MINIMIZED or WINDOW-RESTORED event applied to a child window has no effect on its parent or ancestor windows.
- Resizing or changing the position of a window has no affect on the size or position of any descendants or ancestors of that window.
- The following attributes have the Unknown value (?) until the window is realized:
  - FULL-HEIGHT-CHARS attribute
  - FULL-HEIGHT-PIXELS attribute
  - FULL-WIDTH-CHARS attribute
  - FULL-WIDTH-PIXELS attribute
- If you embed the window in an OpenEdge .NET form, by setting the form's EmbeddedWindow property to the ABL window handle, some attributes and methods of the window are either ignored or change function. For more information, see the EmbeddedWindow property reference entry.

See also

CREATE widget statement, DIALOG-BOX widget

5

Handle Reference

This section contains reference entries that describe the ABL handle-based objects (other than widgets). Handle-based objects represent built-in object types in ABL that you can reference using object handles.

These objects include widgets, which encapsulate user interface capabilities, and other object types that provide access to a variety of ABL session capabilities. Object handles are essentially addresses that provide access to handle-based objects in memory. These handles allow you to access attributes and methods that ABL defines for each object that you can use to interact with an object's state and behavior.

Note: Handle-based objects are conceptually analogous to but different from class-based objects. The types and capabilities of handle-based objects are defined entirely by ABL. However class-based objects represent instances of classes whose capabilities you can define as user-defined data types. For more information on class-based objects, see the CLASS statement reference entry.

You can define static or create dynamic instances of many handle-based object types using appropriate DEFINE and CREATE statements. You can reference a static instance by the defined object name or its handle, and you can reference a dynamic instance only by its handle. ABL also automatically creates instances of certain other handle-based object types in various ways, including system objects that you can access using an appropriate keyword-defined system handle.

Thus, ABL makes object handles available in two ways:

- Directly, as system handle values. A system handle is an ABL keyword that evaluates to an object handle whose object type is implied by the keyword. For example, the CURRENT-WINDOW system handle is a handle to a particular window widget. To access the attributes and methods of a system handle, you can

use the keyword directly, or you can assign the keyword value to a HANDLE variable and use the variable to reference the attributes and methods:

```
DEFINE VARIABLE hHandle AS HANDLE NO-UNDO.

hHandle = THIS-PROCEDURE.          /* Procedure object handle */

DISPLAY THIS-PROCEDURE:GET-SIGNATURE(""). /* Displays the same */
DISPLAY hHandle:GET-SIGNATURE(""). /* Signature as this statement. */
```

- Indirectly, as values output from various CREATE and other ABL statements, including other handle attributes and methods. You can access the attributes and methods of any object by assigning its handle to a HANDLE variable, which can hold handle values of any type. You can then use this variable to reference the attributes and methods of the object:

```
DEFINE VARIABLE hHandle AS HANDLE NO-UNDO.

CREATE SERVER hHandle. /* Create a handle to a server object */
hHandle:CONNECT().     /* Execute a server object method */
```

The reference entries throughout this section include both system handles and handles for object types not necessarily referenced using system handles. Each system handle is listed by its keyword (all upper case, for example: SESSION system handle), and each handle for other object types is listed by its object type (upper and lower case, for example: Server object handle).

Each entry lists the attributes and methods supported by the handle or refers you to a more general entry with the same list. For example, the attributes and methods of the CURRENT-WINDOW system handle appear under the WINDOW widget entry. Widgets share a common set of user interface capabilities as well as capabilities unique to each widget type. For a list of the attributes and methods supported by each widget, see the Widget Reference.

For more information on how to use the list attributes and methods for all handle-based object types, see the Handle Attributes and Methods Reference, which includes a complete reference entry for each attribute and method.

Note: You may consider an object handle to be supported for all interfaces, on all operating systems, and for SpeedScript unless otherwise indicated in the reference entry.

The following descriptions refer to both compile-time and run-time behavior, features that the language generally supports and determines at compile time and actions directed by using these features at run time. When describing compile-time features or actions, this section references ABL or the ABL compiler. When describing ABL-directed actions taken at run time, this section references the ABL Virtual Machine (AVM).

ACTIVE-WINDOW system handle

A handle to the last ABL application window to receive an ENTRY event. You cannot set the ACTIVE-WINDOW handle, but you can read and write values for the attributes of the ACTIVE-WINDOW.

Syntax

```
ACTIVE-WINDOW [ :attribute ]
```

attribute

An attribute of the ACTIVE-WINDOW.

Attributes

The ACTIVE-WINDOW handle has all the attributes of a window widget.

Notes

- In a character interface, the ACTIVE-WINDOW, CURRENT-WINDOW, and DEFAULT-WINDOW handles return the handle of the static window for the current ABL session.
- The initial value of the ACTIVE-WINDOW handle is the CURRENT-WINDOW handle.
- The ACTIVE-WINDOW handle monitors the active window in the ABL session only. It does not monitor the active window for the window system. Accessing a non-ABL window does not affect the state of the ACTIVE-WINDOW handle.
- You can enable or disable the current window by changing ACTIVE-WINDOW:SENSITIVE.
- You can set the menu bar for the active window by assigning the handle of a menu bar to ACTIVE-WINDOW:MENUBAR.

See also

CURRENT-WINDOW system handle, DEFAULT-WINDOW system handle

Asynchronous request object handle

Maintains the status of an asynchronous request running on an OpenEdge application server or Web service.

Syntax

```
async-request-handle [ : attribute ]
```

async-request-handle

A handle variable that references an asynchronous request object. This object is instantiated when you execute an asynchronous remote procedure using the RUN statement specified with the ASYNCHRONOUS option. You can get the handle value by doing one of the following:

- Use the ASYNCHRONOUS SET option on the same RUN statement that instantiates the asynchronous request.
- Reference the LAST-ASYNC-REQUEST attribute on the server handle for the OpenEdge application server where the request is running. To ensure that you are referencing a specific request, you must reference the attribute after the associated RUN statement executes and before

you instantiate another asynchronous request on the same OpenEdge application server connection.

- You can also locate the asynchronous request handle by walking the chain between the FIRST-ASYNC-REQUEST and LAST-ASYNC-REQUEST attributes of the associated server handle. Search on the PROCEDURE-NAME attribute of each request handle to identify the specific request.

attribute

An attribute of the asynchronous request handle.

Attributes

CANCELLED attribute	COMPLETE attribute	ERROR attribute
ERROR-OBJECT attribute	EVENT-HANDLER attribute	EVENT-HANDLER-OBJECT attribute
EVENT-PROCEDURE attribute	EVENT-PROCEDURE-CONTEXT attribute	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	NAME attribute	NEXT-SIBLING attribute
PERSISTENT-PROCEDURE attribute	PREV-SIBLING attribute	PRIVATE-DATA attribute
PROCEDURE-NAME attribute	QUIT attribute	REQUEST-INFO attribute
RESPONSE-INFO attribute	SERVER attribute	STOP attribute
STOP-OBJECT attribute	TYPE attribute	

Events

PROCEDURE-COMPLETE	–

Notes

- When the OpenEdge application server completes and returns the results of the asynchronous request associated with this handle, the client application that executed the request receives the PROCEDURE-COMPLETE event. This event triggers execution of the associated event procedure or method (if specified) in the context of an input-blocking statement, such as the WAIT-FOR statement, UPDATE statement, or a PROCESS EVENTS statement.
- You can access this handle anywhere in the client application that executes the associated request. However, it is especially useful for reference in the event procedure or method specified for the asynchronous request. In the associated event procedure or method, you can access this handle as the value of the SELF system handle.

- When checking for ERROR and STOP conditions raised from the application server in a PROCEDURE-COMPLETE event procedure or method, check the ERROR-OBJECT and STOP-OBJECT attributes (instead of the STOP and ERROR error attributes) to handle these conditions. For example:

```
IF VALID-OBJECT(SELF:ERROR-OBJECT) THEN DO:
    // Perhaps display the error
    MESSAGE SELF:ERROR-OBJECT:GetMessage(1) VIEW-AS ALERT-BOX.
END.

/* That's probably sufficient unless you are using STOP-AFTER.
   It could also possibly be LockConflict, though that would
   normally be handled on the server side. If both are
   possible, check the type to know what it is. */
ELSE IF VALID-OBJECT(SELF:STOP-OBJECT) THEN DO:
    IF TYPE-OF(SELF:STOP-OBJECT, Progress.Lang.StopAfter) THEN
        . . .
END.
```

  Note that references to all objects that implement the `Progress.Lang.Error` interface are returned using the ERROR-OBJECT attribute, including `Progress.Lang.StopError`. The STOP-OBJECT attribute returns only stop object references for `Progress.Lang.Stop` or one of its subclasses, such as `Progress.Lang.StopAfter` in the example.

- The EVENT-PROCEDURE and EVENT-PROCEDURE-CONTEXT attributes will return the Unknown value (?) if the EVENT-HANDLER-CONTEXT is specified as an object.

See also

RUN statement, Server object handle, SELF system handle, WAIT-FOR statement (ABL only)

AUDIT-CONTROL system handle

A handle to the audit control settings for managing application auditing context and events for the current ABL session.

Syntax

```
AUDIT-CONTROL [ :attribute | :method ]
```

attribute

An attribute of the AUDIT-CONTROL system handle.

method

A method of the AUDIT-CONTROL system handle.

Attributes

APPL-CONTEXT-ID attribute	EVENT-GROUP-ID attribute

HANDLE attribute	INSTANTIATING-PROCEDURE attribute
TYPE attribute	–

Methods

BEGIN-EVENT-GROUP( ) method	CLEAR-APPL-CONTEXT( ) method
END-EVENT-GROUP( ) method	LOG-AUDIT-EVENT( ) method
SET-APPL-CONTEXT( ) method	–

See also

AUDIT-POLICY system handle, Client-principal object handle, SECURITY-POLICY system handle

AUDIT-POLICY system handle

A handle that lets you update current audit policy settings for processing audit events and securing audit data for an audit-enabled database.

Syntax

```
AUDIT-POLICY [ :method ]
```

method

A method of the AUDIT-POLICY system handle.

Methods

ENCRYPT-AUDIT-MAC-KEY( ) method	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	REFRESH-AUDIT-POLICY( ) method
TYPE attribute	–

For information about audit-enabling a database, or creating and activating an audit policy for a database, see *Learn about Security and Auditing*.

See also

AUDIT-CONTROL system handle, AUDIT-ENABLED function, Client-principal object handle, SECURITY-POLICY system handle

Buffer object handle

A handle to a buffer object, corresponding to an underlying ABL buffer, which can be static or dynamic. An example of a static underlying buffer is one you define at compile time by using the DEFINE BUFFER statement, or by implicitly referencing a table in an ABL construct such as `Customer.CustNum`. An example of a dynamic underlying buffer is one you create at run time with the CREATE BUFFER statement.

Syntax

```
buffer-handle [ :attribute | :method ]
```

buffer-handle

An item of type HANDLE representing a handle to a buffer object.

attribute

An attribute of the buffer object.

method

A method of the buffer object.

Attributes

ADM-DATA attribute	AFTER-BUFFER attribute	AFTER-ROWID attribute
AMBIGUOUS attribute	ATTACHED-PAIRLIST attribute	AUTO-DELETE attribute
AUTO-SYNCHRONIZE attribute	AVAILABLE attribute	BATCH-SIZE attribute
BEFORE-BUFFER attribute	BEFORE-ROWID attribute	BUFFER-GROUP-ID attribute
BUFFER-GROUP-NAME attribute	BUFFER-PARTITION-ID attribute	BUFFER-TENANT-ID attribute
BUFFER-TENANT-NAME attribute	CAN-CREATE attribute	CAN-DELETE attribute
CAN-READ attribute	CAN-WRITE attribute	CRC-VALUE attribute
CURRENT-CHANGED attribute	CURRENT-ITERATION attribute (Data Objects)	DATASET attribute
DATA-SOURCE attribute	DATA-SOURCE-COMPLETE-MAP attribute	DATA-SOURCE-MODIFIED attribute
DATA-SOURCE-ROWID attribute	DBNAME attribute	DYNAMIC attribute
ERROR attribute	ERROR-STRING attribute	FILL-MODE attribute

HANDLE attribute	HAS-LOBS attribute	INSTANTIATING-PROCEDURE attribute
IS-MULTI-TENANT attribute	IS-PARTITIONED attribute	KEYS attribute
LAST-BATCH attribute	LOCKED attribute	NAME attribute
NAMESPACE-PREFIX attribute	NAMESPACE-URI attribute	NEW attribute
NEXT-SIBLING attribute	NUM-CHILD-RELATIONS attribute	NUM-FIELDS attribute
NUM-ITERATIONS attribute (data objects)	NUM-REFERENCES attribute	ORIGIN-ROWID attribute
PARENT-RELATION attribute	PRIMARY attribute	PRIVATE-DATA attribute
QUERY attribute	RECID attribute	RECORD-LENGTH attribute
REJECTED attribute	ROWID attribute	ROW-STATE attribute
SERIALIZE-NAME attribute	TABLE attribute	TABLE-HANDLE attribute
TABLE-NUMBER attribute	TYPE attribute	UNIQUE-ID attribute
XML-NODE-NAME attribute	–	–

Methods

ACCEPT-CHANGES( ) method	ACCEPT-ROW-CHANGES( ) method
APPLY-CALLBACK( ) method	ATTACH-DATA-SOURCE( ) method
BUFFER-COMPARE( ) method	BUFFER-COPY( ) method
BUFFER-CREATE( ) method	BUFFER-DELETE( ) method
BUFFER-FIELD( ) method	BUFFER-RELEASE( ) method
BUFFER-VALIDATE( ) method	COPY-TEMP-TABLE( ) method
DETACH-DATA-SOURCE( ) method	DISABLE-DUMP-TRIGGERS( ) method
DISABLE-LOAD-TRIGGERS( ) method	EMPTY-TEMP-TABLE( ) method
FILL( ) method	FIND-BY-ROWID( ) method
FIND-CURRENT( ) method	FIND-FIRST( ) method
FIND-LAST( ) method	FIND-UNIQUE( ) method
GET-CALLBACK-PROC-CONTEXT( ) method	GET-CALLBACK-PROC-NAME( ) method

GET-CHANGES( ) method	GET-CHILD-RELATION( ) method
GET-ITERATION( ) method (Data Objects)	INDEX-INFORMATION( ) method
MARK-NEW( ) method	MARK-ROW-STATE( ) method
MERGE-CHANGES( ) method	MERGE-ROW-CHANGES( ) method
RAW-TRANSFER( ) method	READ-JSON( ) method
READ-XML( ) method	READ-XMLSCHEMA( ) method
REJECT-CHANGES( ) method	REJECT-ROW-CHANGES( ) method
SAVE-ROW-CHANGES( ) method	SERIALIZE-ROW( ) method
SET-CALLBACK( ) method	SET-CALLBACK-PROCEDURE( ) method
SYNCHRONIZE( ) method	WRITE-JSON( ) method
WRITE-XML( ) method	WRITE-XMLSCHEMA( ) method

Events

AFTER-FILL event	AFTER-ROW-FILL event
BEFORE-FILL event	BEFORE-ROW-FILL event
FIND-FAILED event	OFF-END event
ROW-CREATE event	ROW-DELETE event
SYNCHRONIZE event	–

For information on these events, see the ProDataSet events.

See also

Buffer-field object handle, CREATE BUFFER statement, DEFINE BUFFER statement, ProDataSet object handle, Query object handle, Temp-table object handle

Buffer-field object handle

A handle to a buffer-field object. Buffer-field objects let you examine and modify the fields of a record. They also let you examine the schema properties of the field.

Syntax

```
buffer-field-handle [ :attribute ]
```

buffer-field-handle

A handle to a buffer-field object.

attribute

An attribute of the buffer-field object.

Attributes

ADM-DATA attribute	AVAILABLE attribute	BUFFER-HANDLE attribute
BUFFER-NAME attribute	BUFFER-VALUE( ) method	CAN-READ attribute
CAN-WRITE attribute	CASE-SENSITIVE attribute	COLUMN-LABEL attribute
DATA-TYPE attribute	DBNAME attribute	DECIMALS attribute
DEFAULT-STRING attribute	DEFAULT-VALUE attribute	EXTENT attribute
FORMAT attribute	HANDLE attribute	HELP attribute
INITIAL attribute	INSTANTIATING-PROCEDURE attribute	KEY attribute
LABEL attribute	LITERAL-QUESTION attribute	MANDATORY attribute
NAME attribute	POSITION attribute	PRIVATE-DATA attribute
READ-ONLY attribute	SERIALIZE-HIDDEN attribute	SERIALIZE-NAME attribute
STRING-VALUE( ) method	TABLE attribute	TYPE attribute
UNIQUE-ID attribute	VALIDATE-EXPRESSION attribute	VALIDATE-MESSAGE attribute
VIEW-AS attribute	WIDTH-CHARS attribute	XML-DATA-TYPE attribute
XML-NODE-NAME attribute	XML-NODE-TYPE attribute	–

See also

Buffer object handle, Query object handle

Call object handle

A handle to a call object allows you to do the following dynamically:

- Invoke an external procedure, internal procedure, or user-defined function
- Invoke a Windows DLL routine or Unix shared library routine
- Get or set an attribute on a handle-based object, such as a buffer or socket
- Run a method on a handle-based object

Syntax

```
call-object-handle [ :attribute | :method ]
```

call-object-handle

The handle to a call object.

attribute

An attribute of the call object.

method

A method of the call object. The methods let you set parameters, reset attributes to their default values, and invoke the call object.

Attributes

ADM-DATA attribute	ASYNCHRONOUS attribute	ASYNC-REQUEST-HANDLE attribute
CALL-NAME attribute	CALL-TYPE attribute	EVENT-PROCEDURE attribute
EVENT-PROCEDURE-CONTEXT attribute	HANDLE attribute	IN-HANDLE attribute
INSTANTIATING-PROCEDURE attribute	IS-PARAMETER-SET attribute	LIBRARY attribute
LIBRARY-CALLING-CONVENTION attribute	NUM-PARAMETERS attribute	ORDINAL attribute
PERSISTENT attribute	PRIVATE-DATA attribute	PROCEDURE-TYPE attribute
RETURN-VALUE attribute	RETURN-VALUE-DATA-TYPE attribute	RETURN-VALUE-DLL-TYPE attribute
SERVER attribute	THREAD-SAFE attribute	TYPE attribute

Methods

CLEAR( ) method (Call Object Handle)	INVOKE( ) method (Handle)
SET-PARAMETER( ) method (Handle)	–

Examples

The following fragment dynamically invokes an external procedure non-persistently:

```
DEFINE VARIABLE hCall AS HANDLE NO-UNDO.

CREATE CALL hCall.

/* Invoke hello.p non-persistently */
hCall:CALL-NAME      = "hello.p".
/* Sets CALL-TYPE to the default */
hCall:CALL-TYPE  = PROCEDURE-CALL-TYPE
hCall:NUM-PARAMETERS = 1.
hCall:SET-PARAMETER(1, "CHARACTER", "INPUT", "HELLO WORLD").
hCall:INVOKE.

/* Clean up */
DELETE OBJECT hCall.
```

The following fragment dynamically invokes an external procedure persistently, then dynamically invokes one of its internal procedures:

```
DEFINE VARIABLE hCall AS HANDLE NO-UNDO.

CREATE CALL hCall.

/* Invoke persis.p persistently */
hCall:CALL-NAME  = "persis.p".
hCall:CALL-TYPE  = PROCEDURE-CALL-TYPE
hCall:PERSISTENT = TRUE.

/* IN-HANDLE automatically set to the handle of the persistent procedure */
hCall:INVOKE.

/* Invoke internal-persis-proc in persis.p */
hCall:CALL-NAME      = "internal-persis-proc".
hCall:NUM-PARAMETERS = 1.
hCall:SET-PARAMETER(1, "INTEGER", "INPUT", 333).
hCall:INVOKE.

/* Clean up */
DELETE PROCEDURE hCall:IN-HANDLE.
DELETE OBJECT hCall.
```

As an alternative to setting the PERSISTENT attribute, a procedure can be run persistently by setting the PROCEDURE-TYPE to "PERSISTENT". Regardless of which attribute is used, the other will automatically be set.

The following fragment dynamically invokes an external procedure as single-run, then dynamically invokes one of its internal procedures:

```
DEFINE VARIABLE hCall AS HANDLE NO-UNDO.

CREATE CALL hCall.

/* Invoke single.p as a single-run procedure*/
hCall:CALL-NAME  = "single.p".
hCall:CALL-TYPE  = PROCEDURE-CALL-TYPE
hCall:PROCEDURE-TYPE = SINGLE-RUN.

/* IN-HANDLE automatically set to the handle of the single-run procedure */
hCall:INVOKE.

/* Invoke internal-single-proc in single.p */
hCall:CALL-NAME      = "internal-single-proc".
hCall:NUM-PARAMETERS = 1.
hCall:SET-PARAMETER(1, "INTEGER", "INPUT", 333).
hCall:INVOKE.

/* Clean up */
DELETE PROCEDURE hCall:IN-HANDLE.
DELETE OBJECT hCall.
```

The previous fragment could also be used to dynamically invoke an external procedure as singleton by changing PROCEDURE-TYPE to "SINGLETON".

The following fragment uses a single call object handle multiple times:

```
DEFINE VARIABLE hCall AS HANDLE NO-UNDO.

CREATE CALL hCall.

/* Invoke hello.p non-persistently */
ASSIGN
  hCall:CALL-NAME      = "hello.p"
  hCall:CALL-TYPE  = PROCEDURE-CALL-TYPE
  hCall:NUM-PARAMETERS = 1.

hCall:SET-PARAMETER(1, "CHARACTER", "INPUT", "HELLO WORLD").
hCall:INVOKE.

/* Reset the call object handle */
hCall:CLEAR.

/* Invoke persis.p persistently */
ASSIGN
  hCall:CALL-NAME  = "persis.p"
  hCall:CALL-TYPE  = PROCEDURE-CALL-TYPE
  hCall:PERSISTENT = TRUE.

/* IN-HANDLE automatically set to the handle of the persistent procedure */
hCall:INVOKE.

/* Invoke internal-persis-proc in persis.p */
ASSIGN
  hCall:CALL-NAME      = "internal-persis-proc"
  hCall:CALL-TYPE  = PROCEDURE-CALL-TYPE
  hCall:NUM-PARAMETERS = 1.

hCall:SET-PARAMETER( 1, "INTEGER", "INPUT", 333).
hCall:INVOKE.

/* Clean up */
DELETE PROCEDURE hCall:IN-HANDLE.
DELETE OBJECT hCall.
```

The following fragment gets an attribute:

```
/* Get title of frame */
ASSIGN
  hCall:IN-HANDLE = myframe_handle
  hCall:CALL-TYPE = GET-ATTR-CALL-TYPE
  hCall:CALL-NAME = "TITLE".

hCall:INVOKE.

Mytitle = hCall:RETURN-VALUE.
```

The following fragment sets an attribute:

```
/* Set SESSION:NUMERIC-FORMAT to "european" */
ASSIGN
  hCall:IN-HANDLE      = "session"
  hCall:CALL-TYPE      = SET-ATTR-CALL-TYPE
  hCall:CALL-NAME      = "numeric-format"
  hCall:NUM-PARAMETERS = 1.

hCall:SET-PARAMETER( 1, "CHAR", "INPUT", "european").
hCall:INVOKE.
```

The following fragment drives the call object's INVOKE( ) method from a TEMP-TABLE:

```
/* Suppose hRuntt is a temp-table that has one record with the following    fields:
   parm_1
   parm_2
   ...
   parm_n
   run-name
   nparms
   datatypes, extent nparms
   iomodes, extent nparms */
DEFINE INPUT PARAMETER TABLE-HANDLE hRuntt.

DEFINE VARIABLE hDtypes  AS HANDLE  NO-UNDO.
DEFINE VARIABLE hIOmodes AS HANDLE  NO-UNDO.
DEFINE VARIABLE hCall    AS HANDLE  NO-UNDO.
DEFINE VARIABLE ix       AS INTEGER NO-UNDO.

ASSIGN
  hDtypes  = hRuntt:BUFFER-FIELD("datatypes")
  hIOmodes = hRuntt:BUFFER-FIELD("iOmodes").

hRuntt:FIND-FIRST.

CREATE CALL hCall.
ASSIGN
  hCall:CALL-NAME      = hRuntt:BUFFER-FIELD("run-name"):BUFFER-VALUE
  hCall:NUM-PARAMETERS = hRuntt:BUFFER-FIELD("nparms"):BUFFER-VALUE.

DO ix = 1 to hCall:NUM-PARAMETERS:
  hCall:SET-PARAMETER(ix, hDtypes:BUFFER-VALUE(ix),
    hIOmodes:BUFFER-VALUE(ix), hRuntt:BUFFER-FIELD(ix):BUFFER-VALUE).
END.

hCall:INVOKE.
DELETE OBJECT hCall.
/* If there are output parms, get values from hRuntt:BUFFER-FIELD(ix) */
```

The following fragment implements an ABL function, `sleep`, which causes the AVM to sleep for a specified number of milliseconds:

```
FUNCTION sleep RETURNS INTEGER (INPUT msecs AS INTEGER):
  DEFINE VARIABLE cFunction AS CHARACTER NO-UNDO INITIAL "sleep".
  DEFINE VARIABLE cLibrary  AS CHARACTER NO-UNDO INITIAL "libc.so.1".
  DEFINE VARIABLE hCall     AS HANDLE    NO-UNDO.

  CREATE CALL hCall.
  ASSIGN
    cLibrary             = "kernel32.dll" WHEN OPSYS = "WIN32"
    cFunction            = "Sleep" WHEN OPSYS = "WIN32"
    hCall:CALL-NAME      = cFunction
    hCall:LIBRARY        = cLibrary
    hCall:CALL-TYPE      = DLL-CALL-TYPE
    hCall:NUM-PARAMETERS = 1.

  hCall:SET-PARAMETER(1, "LONG", "INPUT", msecs).
  hCall:INVOKE( ).

  DELETE OBJECT hCall.
  RETURN msecs.
END FUNCTION.
```

Note that the code checks to determine on which OS it is running, and invokes the appropriate Windows DLL or UNIX shared library.

Notes

- Invoking logic dynamically requires many more lines of code and is less efficient than invoking it statically. You typically use the call object when you cannot use the RUN statement, the DYNAMIC-FUNCTION() function, or *`widget:attribute`* or *`widget:method`* syntax, as in the following situations:
  - To invoke an internal or external procedure whose **calling sequence** (number of parameters and the data type of each) is unknown at compile time.

    Note: If only the **name** of the procedure is unknown at compile time, use the RUN statement with the VALUE option—and avoid the call object altogether.

  - To invoke a function whose calling sequence is unknown at compile time.

    Note: If only the **name** of the function is unknown at compile time, use the DYNAMIC-FUNCTION() function—and avoid the call object altogether.

  - To reference a widget attribute or method whose name is unknown at compile time.
  - To invoke a Windows DLL routine or Unix shared library routine when:
    - The number of parameters and their data type is only known at run time
    - The routine exists in both a Windows DLL and a Unix shared library
    - The routine has a variable number of parameters

  If you already know the name of the attribute or procedure, you know its syntax, since the name implies certain syntax. And if you know the syntax, you know the calling sequence, since the syntax defines the calling sequence. And if you know the calling sequence, you can use *`widget:attribute`* or *`widget:method`* syntax—and avoid the call object altogether.

- To create a call object, use the following syntax:

```
CREATE object-handle [ IN widget-pool ]
```

  For example:

```
CREATE CALL hCall.
```

  Note: Unlike most ABL objects, the call object, by default, goes into the SESSION widget pool, not into the closest unnamed widget pool.

- To delete a call object, use the following syntax:

```
DELETE OBJECT handle.
```

  For example:

```
DELETE OBJECT hCall.
```

  Since the call object, by default, goes into the SESSION widget pool, not into the closest unnamed widget pool, to delete a call object created when the IN widget-pool option is not used, use the `DELETE OBJECT handle` syntax explicitly.

- Setting the PROCEDURE-TYPE attribute to "`PERSISTENT`" will automatically set the PERSISTENT attribute to TRUE. Likewise, setting the PERSISTENT attribute to TRUE will automatically set PROCEDURE-TYPE to "`PERSISTENT`". Setting the PROCEDURE-TYPE and PERSISTENT attributes to conflicting values, e.g., PERSISTENT as TRUE and PROCEDURE-TYPE as "`SINGLE-RUN`", will result in a run-time error.

See also

RUN statement

Client-principal object handle

A handle to a client-principal object. Each client-principal object is an ABL security token. This security token contains user credentials that are used to establish a user identity for an ABL session or database connection, and additional information related to that identity. Once an identity is established, it can be used to authorize such actions as accessing resources (run-time permissions checking) and providing an auditing identity, among other things. For a multi-tenant database connection, establishing its identity also establishes the user's database tenancy.

You can use a client-principal to establish identity in one of two ways, depending on its state. If the object is unsealed (required attributes can be modified), you can set the identity that the security token asserts (through *user authentication*) for an ABL session or database connection by authenticating this identity against specified user accounts and creating a login session for the authenticated user, which also seals the object against any changes to its identity.

If the object is sealed, you can set the identity that the security token represents (through *single sign-on*, or *SSO*) for an ABL session or database connection by validating the object's existing identity and login session for use by the particular ABL session or database connection.

While the basic states of an ABL security token are unsealed or sealed, there are actually several different states that a client-principal object can represent. You can create an ABL security token at run time using the CREATE CLIENT-PRINCIPAL statement. For more information on the states and uses of a client-principal object, see the Notes of this reference entry.

Syntax

```
client-principal-handle [ :attribute | :method ]
```

client-principal-handle

A variable of type HANDLE that references a client-principal object.

attribute

An attribute of the client-principal object handle.

method

A method of the client-principal object handle.

Attributes

AUDIT-EVENT-CONTEXT attribute	CLIENT-TTY attribute
CLIENT-WORKSTATION attribute	DB-LIST attribute
DOMAIN-DESCRIPTION attribute	DOMAIN-NAME attribute
DOMAIN-TYPE attribute	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	LOGIN-EXPIRATION-TIMESTAMP attribute
LOGIN-HOST attribute	LOGIN-STATE attribute
PRIMARY-PASSPHRASE attribute	QUALIFIED-USER-ID attribute
ROLES attribute	SEAL-TIMESTAMP attribute
SESSION-ID attribute	STATE-DETAIL attribute
TYPE attribute	USER-ID attribute

Methods

AUTHENTICATION-FAILED( ) method	EXPORT-PRINCIPAL( ) method
GET-PROPERTY( ) method	IMPORT-PRINCIPAL( ) method
INITIALIZE( ) method	LIST-PROPERTY-NAMES( ) method
LOGOUT( ) method	SEAL( ) method
SET-PROPERTY( ) method	TENANT-ID( ) method
TENANT-NAME( ) method	VALIDATE-SEAL( ) method

Notes

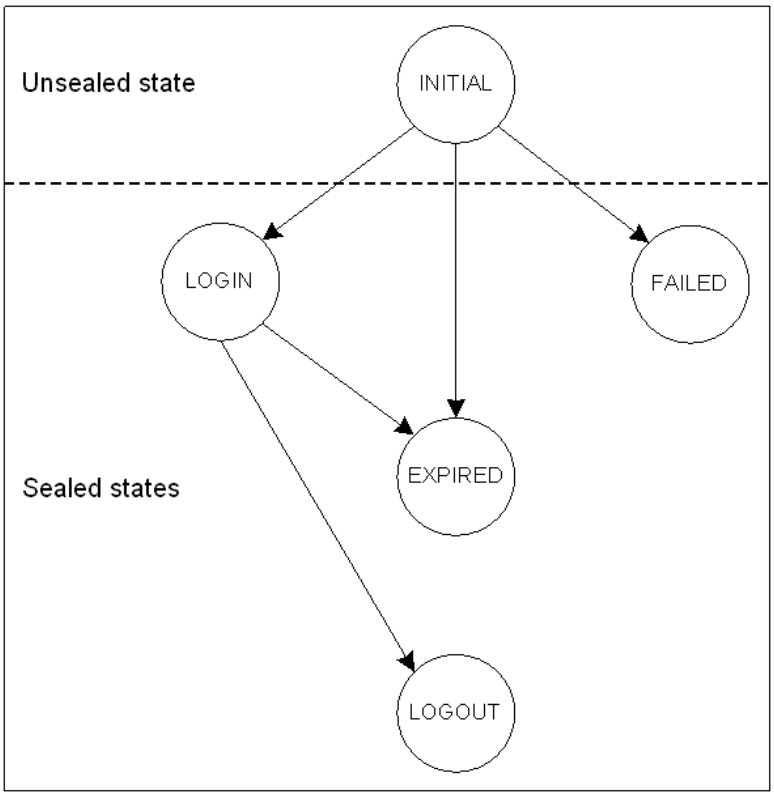
- You can set the values of writable attributes and invoke methods (such as SET-PROPERTY( )) that write data values to a client-principal object only while the object is not sealed (unsealed). An *unsealed* security token is one that has not yet been used to authenticate a user identity and is in the INITIAL state. A *sealed* security token is one that is in a state other than the INITIAL state. For more information on the possible states of a security token, see the LOGIN-STATE attribute entry.
- To set the user identity for an ABL session or OpenEdge database connection, you can use a client-principal object together with either the SECURITY-POLICY:SET-CLIENT( ) method or the SET-DB-CLIENT function in a user authentication or SSO operation. The SETUSERID function also authenticates and sets the user identity for an OpenEdge database connection and creates a security token to store this identity. You can have only one active identity set for each ABL session or database connection at any given point in time, and the identity of an ABL session can be different from the identities of any and all of its database connections. You can also retrieve the existing security token as a client-principal object for any ABL session or database connection identity using the SECURITY-POLICY:GET-CLIENT( ) method or GET-DB-CLIENT function.

  Note: The SETUSERID function provides limited support for user authentication and no support for SSO. It is supported for backward compatibility. For the most complete user authentication and SSO support, use the SET-DB-CLIENT function or the SET-CLIENT( ) method, instead.

- A client-principal object can be used to establish identity through user authentication or SSO, depending on its login state (value of the LOGIN-STATE attribute). As long as the client-principal is unsealed in the INITIAL state, it can establish identity through user authentication. Once it is sealed in the LOGIN state, it can establish identity through SSO. If it is sealed in the LOGOUT, EXPIRED, or FAILED state, you can no

longer use the object to establish identity. The following figure shows the state transitions that can occur during object operations.

Figure 1: Client-principal login state transitions



- The identity to be established by a client-principal must be authenticated or validated (for SSO) using an authentication system associated with the user's assigned OpenEdge security domain. This domain must be enabled and registered in a trusted domain registry, which can be one of the following:
  - An ABL session registry that you construct using methods of the SECURITY-POLICY system handle
  - The local registry provided by a connected OpenEdge RDBMS

  The authentication system associated with the user's domain must support each operation (user authentication or SSO) that you use to establish the user's identity. For more information on authentication systems, OpenEdge security domains, and domain registries, see *Learn about Identity Management*.

- OpenEdge supports two basic forms of user authentication in ABL using a client-principal object:
  - **OpenEdge-performed** — OpenEdge performs all the steps required to authenticate a user's identity. Thus, in one ABL operation OpenEdge:
    1. Submits the user's asserted identity for authentication by the user account system defined for the authentication system associated with the user's domain.
    2. Seals the client-principal object with the user's asserted identity and (if authentication is successful in Step 1) begins a login session for the user.
    3. Assigns the user's identity to an ABL session or database connection.

    Note: Step 1 can include use of an ABL authentication callback to implement or extend this user authentication step. For more information, see the sections on ABL authentication callbacks in *OpenEdge Programming Interfaces*.

- **Application-performed** — The ABL application performs all steps required to authenticate a user's identity. Thus, the ABL application:

  1. Determines and applies the criteria required to authenticate the user identity.
  2. In one ABL operation, seals the client-principal object with the user's asserted identity and (if authentication is successful in Step 1) begins a login session for the user.
  3. In another ABL operation, assigns the user's identity to an ABL session or database connection.

- To establish identity through an OpenEdge-performed user authentication operation, you must first set up a trusted domain registry, which can be the local registry of a connected OpenEdge database or a registry that you create in the ABL session, as described previously in these notes.

  Set the following required attributes of an unsealed client-principal object (some of which have defaults):

  - USER-ID attribute — The user name, which can be set from the user name (unqualified user ID) of the QUALIFIED-USER-ID attribute value.
  - DOMAIN-NAME attribute — The user's domain name, which can be set from the domain name of the QUALIFIED-USER-ID attribute value. The specified domain must be authentication-enabled—that is, an OpenEdge domain that is enabled for run-time access, supports a valid source of user accounts, and is configured with an authentication system that supports OpenEdge-performed user authentication, such as the built-in `_oeusertable` or `_oslocal` authentication system, or a user-defined authentication system that is enabled for authentication using an authentication callback (see *OpenEdge Programming Interfaces*).
  - PRIMARY-PASSPHRASE attribute — The user's password, which can have the default value of blank ("") if no passphrase is required for authentication.
  - SESSION-ID attribute — A unique login session identifier, which must have a value other than blank or the Unknown value (?) that uniquely identifies the login session.

  You can set all of these attributes in one step using the INITIALIZE( ) method. You then execute the SECURITY-POLICY:SET-CLIENT( ) method or SET-DB-CLIENT function to invoke the authentication operation, passing the unsealed client-principal as input. Each ABL operation authenticates the asserted user identity, and if successful, implicitly seals the client-principal using the domain access code that is registered for the user's domain in the trusted domain registry, begins the user's login session, and sets the ABL session, a database connection, or both to the authenticated identity. If successful for the SET-CLIENT( ) method, the operation establishes the identity for the ABL session and for any existing database connection that has not already had an identity established using the SET-DB-CLIENT (or SETUSERID) function. If successful for the SET-DB-CLIENT function, the operation establishes the identity for the specified database connection and locks that database connection against any subsequent change in identity using the SET-CLIENT( ) method. If authentication fails for any reason, including a client-principal that has exceed any setting of its LOGIN-EXPIRATION-TIMESTAMP attribute, the operation implicitly seals the client-principal in the appropriate non-LOGIN state (FAILED or EXPIRED).

  Note: OpenEdge never stores the value of the PRIMARY-PASSPHRASE attribute in a security token or anywhere else. Once passed, together with other required user credentials, to the authentication system for user authentication, OpenEdge destroys all memory of its value.

- To establish identity through an application-performed user authentication operation, you must first set up a trusted domain registry, which can be the local registry of a connected OpenEdge database or a separate registry that you create in the ABL session, as described previously in these notes.

  Your application determines the requirements for authenticating a user identity, which might involve passwords, biometric profiles, and so on. Whatever the requirements, they must result in values you can use to set the following required attributes of an unsealed client-principal object (some of which have defaults):

- USER-ID attribute — The user name, which can be set from the user name (unqualified user ID) of the QUALIFIED-USER-ID attribute value.
- DOMAIN-NAME attribute — The user's domain name, which can be set from the domain name of the QUALIFIED-USER-ID attribute value. The specified domain can be configured for an authentication system that requires application-performed user authentication and supports only OpenEdge-performed SSO, such as `_extsso` or any user-defined authentication system that is not enabled for authentication using an authentication callback.
- SESSION-ID attribute — A unique login session identifier, which must have a value other than blank or the Unknown value (?) that uniquely identifies the login session.

You can set all of these attributes in one operation using the INITIALIZE( ) method. Note that the PRIMARY-PASSPHRASE attribute is not required to initialize the client-principal. Again, your application determines the requirements for authenticating user identity and OpenEdge does not need to know about them.

If the application authentication is successful, you then call the SEAL( ) method on the client-principal object, which explicitly seals the client-principal in the LOGIN state using the domain access code that is registered for the user's domain in the trusted domain registry and begins the user's login session. However, if the client-principal exceeds any setting of its LOGIN-EXPIRATION-TIMESTAMP attribute, this method implicitly seals the object in the EXPIRED state. If the application authentication is not successful for any reason, you can then call the AUTHENTICATION-FAILED( ) method to explicitly seal the client-principal in the FAILED state.

To set the successfully asserted and sealed identity for the current ABL session, a database connection, or both, you pass the sealed client-principal to the SECURITY-POLICY:SET-CLIENT( ) method or SET-DB-CLIENT function, as appropriate. If successful for the SET-CLIENT( ) method, the operation establishes the identity for the ABL session and for any existing database connection that has not already had an identity established using the SET-DB-CLIENT (or SETUSERID) function. If successful for the SET-DB-CLIENT function, the operation establishes the identity for the specified database connection and locks that database connection against any subsequent change in identity using the SET-CLIENT( ) method.

- To establish identity through an SSO operation, you first need an implicitly or explicitly sealed client-principal object that represents the user identity. You then invoke the SECURITY-POLICY:SET-CLIENT( ) method or the SET-DB-CLIENT function with the sealed client-principal object as input. This SSO operation validates the domain specified for the client-principal against the domain's access code in the trusted domain registry. If successful for the SET-CLIENT( ) method, the operation establishes the identity for the ABL session and for any existing database connection that has not already had an identity established using the SET-DB-CLIENT (or SETUSERID) function. If successful for the SET-DB-CLIENT function, the operation establishes the identity for the specified database connection and locks that database connection against any subsequent change in identity using the SET-CLIENT( ) method.
- When any user authentication or SSO operation successfully sets the user identity for a multi-tenant database connection, the operation also establishes the user's database tenant organization as defined by the user's domain. In addition, this operation stores the tenant information for this database connection in the client-principal object, including the database name, tenant name, and a tenant ID, which together uniquely identify the tenant for this particular database connection. This unique tenancy thus distinguishes tenants among connections to multiple multi-tenant databases that share the same user identity for more than one connection. This tenant information can be returned using the DB-LIST attribute, TENANT-ID( ) method, and TENANT-NAME( ) method of the client-principal object for every database connection that has been set to the user identity using this instance of the client-principal object only. This information can be returned for a database that has nothing to do with the current ABL session, because the same client-principal instance can be used to establish identity in multiple ABL sessions where it can accumulate tenant information for multi-tenant database connections across sessions.
- In addition to the invoking the SET-CLIENT( ) method or SET-DB-CLIENT function with an unsealed client-principal, OpenEdge performs a user authentication operation that creates a sealed security token for a specified database connection from asserted user credentials using the SETUSERID function or when

asserting identity with the User ID (-U) and Password (-P) connection parameters to the CONNECT statement and the AVM startup command line. During **any** OpenEdge-performed user authentication operation, OpenEdge assigns the values of the following attributes (if not already set) from the domain configuration before sealing the associated security token:

- AUDIT-EVENT-CONTEXT attribute
- DOMAIN-DESCRIPTION attribute
- DOMAIN-TYPE attribute
- STATE-DETAIL attribute (if the authentication operation fails)

- At any point during a user login session, you can log the user out by invoking the LOGOUT( ) method on the sealed client-principal object (in the LOGIN state) that represents the user's identity. This method thus changes the object's state from LOGIN to LOGOUT.
- You can use the EXPORT-PRINCIPAL( ) method and IMPORT-PRINCIPAL( ) method to transport a sealed or unsealed security token from one ABL session to another, where its identity can be authenticated during login or validated during SSO operations, as appropriate. For an unsealed client-principal that you export outside the current session, you can also encrypt any setting of the PRIMARY-PASSPHRASE attribute. For more information, see the description of how to encrypt passwords in the ENCRYPT-AUDIT-MAC-KEY( ) method entry.
- Certain methods invoked on or using a client-principal object generate audit events. For more information, see the AUDIT-EVENT-CONTEXT attribute entry.

See also

AUDIT-CONTROL system handle, AUDIT-POLICY system handle, CREATE CLIENT-PRINCIPAL statement, SECURITY-POLICY system handle, SET-CLIENT( ) method, SETUSERID function, SET-DB-CLIENT function

CLIPBOARD system handle

A handle to the system clipboard widget. The CLIPBOARD handle allows you to implement interactions that allow the user to transfer data between ABL field-level widgets, or between ABL field-level widgets and the widgets of other applications running on the system. ABL can interpret the data read from or written to the system clipboard widget as a single item or as a group of multiple items. These data transfers are typically invoked as cut, copy, and paste operations.

Syntax

```
CLIPBOARD [ :attribute ]
```

attribute

An attribute of the clipboard widget.

Attributes

AVAILABLE-FORMATS attribute	HANDLE attribute	INSTANTIATING-PROCEDURE attribute

ITEMS-PER-ROW attribute	MULTIPLE attribute	NUM-FORMATS attribute
TYPE attribute	VALUE attribute	–

Examples

The following code fragment implements cut, copy, and paste operations for the EM_Cut, EM_Copy, and EM_Paste items on the EditMenu menu. It uses the FOCUS handle to reference the widget that has the current input focus.

Note that the fragment tests the widget type of the FOCUS widget in two instances: once when EditMenu is opened during the MENU-DROP event to determine what clipboard operations are valid for the widget, and once again when a clipboard operation is chosen from the menu to determine how the operation is executed for the widget. During the MENU-DROP event, if a particular operation is valid for the FOCUS widget the menu item for that operation is enabled. Otherwise, it is disabled.

During the CHOOSE event for a n enabled menu item, the fragment executes the corresponding clipboard operation in a way that accounts for the unique features of the FOCUS widget. For example, the copy operation (EM_Copy) copies the selected text from an editor widget, copies the label text from a radio set item, and copies a composed true or false message for a toggle box. Your own implementation of these operations for the same widgets can be quite different.

For a complete description of this example, see the topics on the system clipboard in *OpenEdge Programming Interfaces*.

```
DEFINE VARIABLE lStat AS LOGICAL NO-UNDO.
. . .
ON MENU-DROP OF MENU EditMenu DO:
  IF FOCUS:TYPE = "EDITOR" THEN DO:
    MENU-ITEM EM_Cut:SENSITIVE IN MENU EditMenu =
      IF LENGTH(FOCUS:SELECTION-TEXT) > 0 THEN TRUE ELSE FALSE.
    MENU-ITEM Em_Copy:SENSITIVE IN MENU EditMenu =
      IF LENGTH(FOCUS:SELECTION-TEXT) > 0 THEN TRUE ELSE FALSE.
    MENU-ITEM EM_Paste:SENSITIVE IN MENU EditMenu =
      IF CLIPBOARD:NUM-FORMATS > 0 THEN TRUE ELSE FALSE.
  END.
  ELSE IF FOCUS:TYPE = "RADIO-SET" OR
          FOCUS:TYPE = "SELECTION-LIST" OR
          FOCUS:TYPE = "SLIDER" OR
          FOCUS:TYPE = "TOGGLE-BOX" THEN DO:
    MENU-ITEM EM_Cut:SENSITIVE IN MENU EditMenu = FALSE.
    MENU-ITEM Em_Copy:SENSITIVE IN MENU EditMenu = TRUE.
    MENU-ITEM Em_Paste:SENSITIVE IN MENU EditMenu = FALSE.
  END.
  ELSE IF FOCUS:TYPE = "FILL-IN" THEN DO:
    MENU-ITEM EM_Cut:SENSITIVE IN MENU EditMenu =
      IF LENGTH(FOCUS:SCREEN-VALUE) > 0 THEN TRUE ELSE FALSE.
    MENU-ITEM Em_Copy:SENSITIVE IN MENU EditMenu =
      IF LENGTH(FOCUS:SCREEN-VALUE) > 0 THEN TRUE ELSE FALSE.
    MENU-ITEM EM_Paste:SENSITIVE IN MENU EDitMenu =
      IF CLIPBOARD:NUM-FORMATS > 0 THEN TRUE ELSE FALSE.
  END.
  ELSE DO:
    MENU-ITEM EM_Cut:SENSITIVE IN MENU EditMenu = FALSE.
    MENU-ITEM EM_Copy:SENSITIVE IN MENU EditMenu = FALSE.
    MENU-ITEM EM_Paste:SENSITIVE IN MENU EditMenu = FALSE.
  END.
END. /* ON MENU-DROP IN EditMenu */

ON CHOOSE OF MENU-ITEM EM_Cut IN MENU EditMenu DO:
  IF FOCUS:TYPE = "EDITOR" THEN DO:
    IF FOCUS:SELECTION-START <> FOCUS:SELECTION-END THEN ASSIGN
      CLIPBOARD:VALUE = FOCUS:SELECTION-TEXT
```

```
      lStat           = FOCUS:REPLACE-SELECTION-TEXT("").
    ELSE ASSIGN
      CLIPBOARD:VALUE    = FOCUS:SCREEN-VALUE
      FOCUS:SCREEN-VALUE = "".
  END.
  ELSE ASSIGN /* For FILL-IN */
    CLIPBOARD:VALUE    = FOCUS:SCREEN-VALUE
    FOCUS:SCREEN-VALUE = "".
END. /* ON CHOOSE OF MENU-ITEM EM_Cut */

ON CHOOSE OF MENU-ITEM EM_Copy IN MENU EditMenu DO:
  IF FOCUS:TYPE = "EDITOR" THEN
    IF FOCUS:SELECTION-START <> FOCUS:SELECTION-END THEN
      CLIPBOARD:VALUE = FOCUS:SELECTION-TEXT.
    ELSE
      CLIPBOARD:VALUE = FOCUS:SCREEN-VALUE.
    ELSE IF FOCUS:TYPE = "RADIO-SET" THEN
      CLIPBOARD:VALUE = ENTRY(LOOKUP(FOCUS:SCREEN-VALUE,
        FOCUS:RADIO-BUTTONS) - 1, FOCUS:RADIO-BUTTONS).
    ELSE IF FOCUS:TYPE = "TOGGLE-BOX" THEN
      IF FOCUS:SCREEN-VALUE = "yes" THEN
        CLIPBOARD:VALUE = FOCUS:LABEL + " selected.".
      ELSE
        CLIPBOARD:VALUE = FOCUS:LABEL + " not selected.".
    ELSE /* For FILL-IN */
       CLIPBOARD:VALUE = FOCUS:SCREEN-VALUE.
END. /* ON CHOOSE OF MENU-ITEM EM_Copy */

ON CHOOSE OF MENU-ITEM EM_Paste IN MENU EditMenu DO:
  IF FOCUS:TYPE = "EDITOR" THEN DO:
    IF FOCUS:SELECTION-START <> FOCUS:SELECTION-END THEN
      lStat = FOCUS:REPLACE-SELECTION-TEXT(CLIPBOARD:VALUE).
    ELSE
      aResult = FOCUS:INSERT-STRING(CLIPBOARD:VALUE).
  END.
  ELSE /* For FILL-IN */
    FOCUS:SCREEN-VALUE = CLIPBOARD:VALUE.
END. /* ON CHOOSE OF MENU-ITEM EM_Paste */
. . .
```

The following `r-clpmul.p` procedure demonstrates interaction with the clipboard using multiple items. The procedure copies out four rows of five numbers to the clipboard. It first displays the clipboard data as a single item, and then as a list of multiple items.

As a further demonstration of how the CLIPBOARD handle works with multiple items, try the following experiment:

1. Run the procedure, and at the pause, paste the result into an edit tool in your window system, such as Notepad in Windows.
2. You may have to select and copy text in the edit tool to activate the system clipboard before running the procedure.
3. Modify the text in the edit tool, leaving at least one tab or newline character, and copy it back to the clipboard from the edit tool.
4. Respond to the pause in the procedure to see how the modified clipboard data is displayed.

r-clpmul.p

```
DEFINE VARIABLE ClipBuffer AS CHARACTER NO-UNDO
  VIEW-AS EDITOR SIZE 60 BY 5.
DEFINE VARIABLE ClipItem   AS CHARACTER NO-UNDO.
DEFINE VARIABLE ix         AS INTEGER   NO-UNDO.

/* Copy rows of integer items to the clipboard, display the clipboard value. */
ASSIGN
  CLIPBOARD:MULTIPLE      = TRUE
  CLIPBOARD:ITEMS-PER-ROW = 5.

REPEAT ix = 1 TO 20:
  CLIPBOARD:VALUE = STRING(ix).
END.

ASSIGN
  CLIPBOARD:MULTIPLE = FALSE
  ClipBuffer         = CLIPBOARD:VALUE.

ENABLE ClipBuffer WITH FRAME A.
DISPLAY SPACE(1) ClipBuffer LABEL "Clipboard Data" WITH FRAME A.
PAUSE.

/* Display each item of the clipboard value. */
ASSIGN
  CLIPBOARD:MULTIPLE = TRUE
  ClipItem           = "".

REPEAT WHILE ClipItem <> ?:
  ClipItem = CLIPBOARD:VALUE.
  IF ClipItem <> ? THEN
    DISPLAY SPACE(1) ClipItem FORMAT "x(16)" LABEL "Clipboard Item"
      WITH DOWN FRAME B.
END.

CLIPBOARD:MULTIPLE = FALSE.
```

Notes

- In character mode environments where there is no system clipboard, ABL supports CLIPBOARD handle operations within a single ABL application. You can cut and paste among fields in one ABL application, but not between one ABL application and another ABL or non-ABL application.
- In graphical window systems, ABL supports CLIPBOARD handle operations using the system clipboard. This allows data transfers among ABL and non-ABL applications as well as within a single ABL application.
- The AVAILABLE-FORMATS attribute returns a comma-delimited string containing the names of the available formats for the data stored in the clipboard. ABL currently supports two formats:
  - **PRO_TEXT** — Specifies the standard text format on your system (CF_TEXT in Windows).
  - **PRO_MULTIPLE** — Specifies that the data in the clipboard contains tab or newline characters, and thus can be read as multiple items.
- The ITEMS-PER-ROW attribute specifies how the AVM writes multiple items to the clipboard. Set the MULTIPLE attribute to TRUE before specifying ITEMS-PER-ROW. Then when you set ITEMS-PER-ROW to any integer value *n* greater than 1, the AVM terminates every *n*th value you assign to the VALUE attribute with a newline character and terminates all other values with a tab character. This formats the output in the clipboard into newline-terminated rows of *n* items separated by tabs. The default value for the ITEMS-PER-ROW attribute is 1.

During a MULTIPLE write, you can set and reset ITEMS-PER-ROW at any time until you set the MULTIPLE attribute to FALSE. When you set the MULTIPLE attribute to FALSE, the AVM uses the current value of ITEMS-PER-ROW to format and flush the data to the clipboard, and resets the ITEMS-PER-ROW attribute to 1.

The value of ITEMS-PER-ROW has no effect when reading data from the clipboard.

- The MULTIPLE attribute specifies whether the AVM reads data from, and writes data to, the clipboard as a single item or as multiple items.

  When you set MULTIPLE to FALSE, the AVM treats all data in the clipboard as a single item. Thus, any character string you assign to the VALUE attribute replaces all data in the clipboard, and whenever you read the VALUE attribute it returns all the data in the clipboard.

  When you set MULTIPLE to TRUE, the AVM treats the data in the clipboard as multiple items separated by tab or newline characters.

  When you set the MULTIPLE attribute to TRUE and write values to the clipboard (assign values to the VALUE attribute), the AVM stores the values in a buffer until you set MULTIPLE to FALSE. At this time the AVM assigns the values to the clipboard separated from each other by tab or newline characters according to the value of the ITEMS-PER-ROW attribute. Note that the clipboard data itself does not change until you set MULTIPLE to FALSE. When you do set MULTIPLE to FALSE, all data previously in the clipboard is replaced by the items you have written.

  When you set the MULTIPLE attribute to TRUE and read values from the clipboard (assign values from the VALUE attribute), each read returns the next item in the clipboard (starting with the first one). After all items have been read, the VALUE attribute returns the Unknown value (?). Setting the MULTIPLE attribute to FALSE and then to TRUE restarts the item pointer to read the first item of data in the clipboard.

  Until you (or another application) write data to the clipboard, changing the value of the MULTIPLE attribute itself has no effect on clipboard contents. It only affects the way you can access the clipboard for reading and writing.

  The default value for the MULTIPLE attribute is FALSE.

- The NUM-FORMATS attribute returns the number of formats available to read data from the clipboard. If no data is in the clipboard, the value is 0. If data is in the clipboard, the value is 1 (for PRO_TEXT) unless there are tab or newline characters in the data, in which case the value is 2 (for both PRO_TEXT and PRO_MULTIPLE).

- The VALUE attribute accesses the data in the clipboard. Reading the VALUE attribute has no effect on the clipboard contents. However, the exact value read or written depends on the setting of the MULTIPLE attribute.

  When the MULTIPLE attribute is FALSE, reading the VALUE attribute returns the current value in the clipboard as a single item. If there is no data in the clipboard, the VALUE attribute returns the Unknown value (?). Writing to the VALUE attribute immediately changes the current value in the clipboard to the value that is written.

  When the MULTIPLE attribute is TRUE, reading the VALUE attribute either references one of the multiple data items in the clipboard, or references the Unknown value (?) if all items have been read or there is no data in the clipboard. Writing to the VALUE attribute buffers each assignment and replaces the current data in the clipboard with the multiple values assigned when the MULTIPLE attribute is set to FALSE. See the previous description of the MULTIPLE attribute for more information.

Note: Windows provides clipboard storage for a maximum of 64K of data.

Assigning the Unknown value (?) to the VALUE attribute has no effect. To write a null item or clear the system clipboard when writing a single item, assign the null string ("") to the VALUE attribute.

- To cut or copy an ABL data item to the clipboard, set the CLIPBOARD:VALUE attribute to the value of the appropriate field or variable. A cut or copy operation replaces all data in the clipboard with the data from the specified ABL field or variable.
- To paste data from the clipboard to an ABL data item, assign the value of the CLIPBOARD:VALUE attribute to the appropriate the field or variable. If there is no data in the clipboard, a paste operation assigns the Unknown value (?) to the data item.
- To implement clipboard operations, use the FOCUS system handle, which identifies the ABL field-level widget that has the current input focus. Depending on the type of widget (for example, EDITOR or RADIO-ITEM) and its input state, you use one of several possible widget attributes as the source or destination for the data. For example, when working with selected text in an editor widget, use the SELECTION-TEXT attribute to cut or copy and the REPLACE-SELECTION-TEXT method to paste, but when working with the value of the entire editor field, use the SCREEN-VALUE attribute for all operations.
- Do not interrupt an ABL clipboard operation with input-blocking statements like UPDATE or WAIT-FOR. In general, make any ABL clipboard cut, copy, or paste operation with the CLIPBOARD handle a one-step operation. Any interruption gives the user an opportunity to access and modify the clipboard from outside ABL, in the middle of the ABL clipboard operation.
- Windows provides default clipboard operations through control keys, whether or not you implement them with the CLIPBOARD handle. These operations are available in editor and fill-in widgets, and are completely compatible with CLIPBOARD handle operations. They are single-item operations without any interaction with the MULTIPLE attribute. They also can occur in the middle of an ABL clipboard operation, if it is interrupted. (See the previous bullet on interrupting ABL clipboard operations.) The operations and control keys to activate them include:
  - **Cut** — **CTRL+X** and **SHIFT+DEL**
  - **Copy** — **CTRL+C** and **CTRL+INS**
  - **Paste** — **CTRL+V** and **SHIFT+INS**
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.
- For more information on implementing clipboard operations with the CLIPBOARD handle, see *OpenEdge Programming Interfaces*.

See also

FOCUS system handle

CODEBASE-LOCATOR system handle
(Windows only; Graphical interfaces only)

A handle to the CODEBASE-LOCATOR object. A CODEBASE-LOCATOR object specifies the location and authentication information for a client application's codebase (that is, an application's files) stored on an application server or a web server. This object allows the WebClient to access application files for download. It also allows WebClient and the client application to share authentication information.

Note: Does not apply to SpeedScript programming.

Syntax

```
CODEBASE-LOCATOR [ :attribute ]
```

attribute

Specifies an attribute of the CODEBASE-LOCATOR handle.

Attributes

APPSERVER-INFO attribute	APPSERVER-PASSWORD attribute	APPSERVER-USERID attribute
END-USER-PROMPT attribute	HANDLE attribute	INSTANTIATING-PROCEDURE attribute
KEEP-CONNECTION-OPEN attribute	KEEP-SECURITY-CACHE attribute	LOCATOR-TYPE attribute
NEEDS-APPSERVER-PROMPT attribute	NEEDS-PROMPT attribute	PERSISTENT-CACHE-DISABLED attribute
SERVER attribute	TYPE attribute	URL attribute
URL-PASSWORD attribute	URL-USERID attribute	–

Notes

- The CODEBASE-LOCATOR handle applies only to WebClient.
- The following attributes represent the security cache: APPSERVER-INFO, APPSERVER-PASSWORD, APPSERVER-USERID, URL-PASSWORD, URL-USERID, and KEEP-SECURITY-CACHE. These attributes are readable and writable.
- WebClient sets the following read-only attributes based on values stored in the application configuration (`.ProwcApp`) file: END-USER-PROMPT, KEEP-CONNECTION-OPEN, LOCATOR-TYPE, NEEDS-APPSERVER-PROMPT, NEEDS-PROMPT, PERSISTENT-CACHE-DISABLED, and URL.
- Valid URL protocols depend on the LOCATOR-TYPE. If LOCATOR-TYPE is "`AppServer`", valid URL protocols include: HTTP, HTTPS, and AppServer. If LOCATOR-TYPE is "`InternetServer`", valid URL protocols include: HTTP, HTTPS, and FILE.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.

See also

CONNECT( ) method (AppServer)

COLOR-TABLE system handle
(Windows only; Graphical interfaces only)

A handle to the current color table.

Note: Does not apply to SpeedScript programming.

Syntax

```
COLOR-TABLE [ :attribute | :method ]
```

attribute

An attribute of the COLOR-TABLE handle.

method

A method of the COLOR-TABLE handle.

Attributes

HANDLE attribute	INSTANTIATING-PROCEDURE attribute	NUM-ENTRIES attribute
TYPE attribute	–	–

Methods

GET-BLUE-VALUE( ) method	GET-DYNAMIC( ) method
GET-GREEN-VALUE( ) method	GET-RED-VALUE( ) method
GET-RGB-VALUE( ) method	SET-BLUE-VALUE( ) method
SET-DYNAMIC( ) method	SET-GREEN-VALUE( ) method
SET-RED-VALUE( ) method	SET-RGB-VALUE( ) method

Example

This procedure sets the number of entries in the color table, makes color ix + 1 dynamic, then sets the red, green, and blue values for this entry:

r-colhan.p

```
DEFINE VARIABLE red   AS INTEGER NO-UNDO.
DEFINE VARIABLE blue  AS INTEGER NO-UNDO INITIAL 127.
DEFINE VARIABLE green AS INTEGER NO-UNDO INITIAL 127.
DEFINE VARIABLE ix    AS INTEGER NO-UNDO.

ix = COLOR-TABLE:NUM-ENTRIES
COLOR-TABLE:NUM-ENTRIES = ix + 1.

COLOR-TABLE:SET-DYNAMIC(ix, TRUE).
COLOR-TABLE:SET-RED-VALUE(ix, red).
COLOR-TABLE:SET-GREEN-VALUE(ix, green).
COLOR-TABLE:SET-BLUE-VALUE(ix, blue).

DISPLAY COLOR-TABLE:GET-RED-VALUE(ix).
DISPLAY COLOR-TABLE:GET-GREEN-VALUE(ix).
DISPLAY COLOR-TABLE:GET-BLUE-VALUE(ix).
```

Note: In this procedure, you can replace the SET-RED-VALUE( ), SET-GREEN-VALUE( ), and SET-BLUE-VALUE( ) methods with the SET-RGB-VALUE( ) method as follows: `COLOR-TABLE:SET-RGB-VALUE(ix, RGB-VALUE(red, green, blue))`.

Notes

- The current color table is the color table in the startup environment or the environment most recently specified in a USE statement.
- To determine the number of entries in the color table, access the NUM-ENTRIES attribute. For character interfaces, the value of this attribute is zero.
- To change the number of entries in the color table, modify the NUM-ENTRIES attribute.
- To let users modify color table entries at run time, display the System Color dialog box by coding the SYSTEM-DIALOG COLOR statement.
- To specify a red, green, or blue value for a dynamic color, supply an integer expression that returns a value between 0 and 255 inclusive.
- To save a color definition from the color table to the current environment, use the PUT-KEY-VALUE statement. To retrieve a color definition from the current environment, use the GET-KEY-VALUE statement.
- The value of COLOR-TABLE:TYPE is "PSEUDO-WIDGET."
- The SET-RGB-VALUE() and GET-RGB-VALUE() methods can be used as an alternative to specifying each individual red, green, and blue color value with the individual SET-RED-VALUE(), SET-GREEN-VALUE(), SET-BLUE-VALUE() methods, and GET-RED-VALUE(), GET-GREEN-VALUE(), and GET-BLUE-VALUE() methods, respectively.
- The SET-RGB-VALUE() and GET-RGB-VALUE() methods to set or retrieve colors are primarily used for Active X controls.
- The index is zero based. For example, the statement `COLOR-TABLE:GET-BLUE-VALUE(2)` returns the color of the 3rd entry.

See also

GET-KEY-VALUE statement, PUT-KEY-VALUE statement, SYSTEM-DIALOG COLOR statement, USE statement

COM-SELF system handle
(Windows only)

A component handle to the ActiveX object (ActiveX control or ActiveX automation object) that generated the event being handled by the currently executing ActiveX event procedure.

Note: Does not apply to SpeedScript programming.

Syntax

```
COM-SELF [ :OCX-property-reference | :OCX-method-reference ]
```

OCX-property-reference | *OCX-method-reference*

A reference to a valid property or method associated with the ActiveX control.

Example

The following code fragment displays the name and position of the ActiveX control that generates a Click event:

```
PROCEDURE ANYWHERE.Click:
  MESSAGE "Clicked control" COM-SELF:Name "at X-position" COM-SELF:Left           "and
 Y-position" COM-SELF:Top VIEW-AS ALERT-BOX.
END PROCEDURE.
```

Notes

- Unlike ABL widget handles that have the HANDLE data type, the component handle returned by COM-SELF has the COM-HANDLE data type.
- You can reference the COM-SELF handle only within an ActiveX control (OCX) event procedure.
- The syntax for referencing ActiveX control properties and methods extends the syntax for referencing widget attributes and methods. For more information, see the Handle Attributes and Methods Reference.

See also

PROCEDURE statement, SELF system handle

COMPILER system handle

A handle to information on a preceding COMPILE statement.

Syntax

```
COMPILER [ :attribute ]
```

attribute

Specifies an attribute of the COMPILER system handle.

Attributes

CLASS-TYPE attribute	ERROR attribute	ERROR-COLUMN attribute
ERROR-ROW attribute	FILE-NAME attribute	FILE-OFFSET attribute
HANDLE attribute	INSTANTIATING-PROCEDURE attribute	MULTI-COMPILE attribute
NUM-MESSAGES attribute	OPTIONS attribute	STOPPED attribute
TYPE attribute	WARNING attribute	–

Methods

GET-COLUMN( ) method	GET-ERROR-COLUMN( ) method
GET-ERROR-ROW( ) method	GET-FILE-NAME( ) method
GET-FILE-OFFSET( ) method	GET-MESSAGE( ) method (Handle)
GET-MESSAGE-TYPE( ) method	GET-NUMBER( ) method
GET-ROW( ) method	–

Example

The input for the example procedure is a comma-separated list of source files. It compiles each of these procedures. If a compilation error occurs, an appropriate message is written to the `compile.msgs` file.

r-cmpchk.p

```
/* Compile a series of source files passed in a comma separated list. */
DEFINE INPUT PARAMETER sources AS CHARACTER NO-UNDO.

DEFINE VARIABLE entry-num AS INTEGER NO-UNDO.

/* If the output file already exists, delete it. If this results in an error,
   ignore the error. */
OS-DELETE "compile.msgs".

DO entry-num = 1 TO NUM-ENTRIES(sources):
  COMPILE VALUE(ENTRY(entry-num, sources)) SAVE.
  IF COMPILER:ERROR THEN DO:
    OUTPUT TO "compile.msgs" APPEND.
    MESSAGE "Compilation error in" COMPILER:FILE-NAME "at line"
       COMPILER:ERROR-ROW "column" COMPILER:ERROR-COL.
    OUTPUT CLOSE.
  END.
END.
```

Notes

- If a compilation is successful, the COMPILER:ERROR attribute is set to FALSE.
- After a COMPILE statement, check the COMPILER:ERROR and COMPILER:WARNING attributes to determine whether the compilation was successful. If the value of ERROR is TRUE, you can use the FILE-NAME to determine in which source file the error occurred. You can use either the ERROR-ROW and ERROR-COLUMN attributes or the FILE-OFFSET attribute to determine where in the source file an error occurred. You can use this information to compose a message to display or write to a log file. To find the specific error and warning messages, check the ERROR-STATUS handle.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.

See also

COMPILE statement

CURRENT-WINDOW system handle

A handle to the default window for the current ABL session. This window is the default parent for all frames, dialog boxes, alert boxes, and messages. Set or examine the attributes of the CURRENT-WINDOW handle to modify or get information on the current default window.

Syntax

```
CURRENT-WINDOW [ :attribute ]
```

attribute

An attribute of the CURRENT-WINDOW.

Attributes

The CURRENT-WINDOW handle has all the attributes of a window widget.

Notes

- The default value of the CURRENT-WINDOW handle is the static session window referenced by the DEFAULT-WINDOW handle. You can change the default window for the current session by assigning the handle of a window to CURRENT WINDOW.
- The IN WINDOW phrase allows you to explicitly assign a window as a parent for a frame, dialog box, alert box, or message.
- In a character interface, the ACTIVE-WINDOW, CURRENT-WINDOW, and DEFAULT-WINDOW handles return the handle of the static window for the current ABL session.
- The CURRENT-WINDOW attribute of a procedure allows you to specify a default window for the procedure block. The CURRENT-WINDOW attribute of a procedure overrides the CURRENT-WINDOW handle for the procedure block.
- You can enable or disable the current window by changing CURRENT-WINDOW:SENSITIVE.
- You can set the menu bar for the current window by assigning the handle of a menu bar to CURRENT-WINDOW:MENUBAR.

- You can make the current window visible or invisible by changing the value of CURRENT-WINDOW:VISIBLE.
- Because CURRENT-WINDOW is the general default parent for ABL frames, you cannot set CURRENT-WINDOW to the handle of a .NET form's shadow window. If allowed, this would make the associated .NET form a default parent for ABL frames, and ABL cannot parent ABL frames to .NET forms. A .NET form's shadow window handle is provided by the ProWinHandle property of the Progress.Windows.Form class, which an OpenEdge .NET class designed for instantiating .NET forms in an ABL session. Therefore, any attempt to set CURRENT-WINDOW to a ProWinHandle property value raises a run-time error.

See also

ACTIVE-WINDOW system handle, DEFAULT-WINDOW system handle

Data-relation object handle

A handle to a data-relation object. A data-relation object defines one relation between a pair of parent and child buffers in a ProDataSet object that have a one-to-one or one-to-many parent-child relationship. A data-relation object identifies the parent and child buffers, and the fields in each buffer that define the primary and foreign key fields of the relation.

A data-relation object can be static or dynamic. A static data-relation object is one you define at compile time as part of the DEFINE DATASET statement using the data-relation option. A dynamic data-relation object is one you create at run time for a dynamic ProDataSet object using the ADD-PARENT-ID-RELATION( ) method. Use this handle to access the data-relation object's attributes.

Note: Does not apply to SpeedScript programming.

Syntax

```
data-relation-handle [ :attribute ]
```

data-relation-handle

A handle variable that references a data-relation object.

attribute

An attribute of the data-relation object.

Attributes

ACTIVE attribute	ADM-DATA attribute	CHILD-BUFFER attribute
CURRENT-QUERY( ) method	FOREIGN-KEY-HIDDEN attribute	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	MAXIMUM-LEVEL attribute	NAME attribute
NESTED attribute	PARENT-BUFFER attribute	PRIVATE-DATA attribute

PARENT-FIELDS-AFTER attribute	PARENT-FIELDS-BEFORE attribute	PARENT-ID-RELATION attribute
QUERY attribute	RECURSIVE attribute	RELATION-FIELDS attribute
REPOSITION attribute	TYPE attribute	WHERE-STRING attribute

Notes

- You cannot define multiple data-relation objects for the same pair of parent and child buffers.
- You cannot delete a data-relation object. When the ProDataSet object is deleted or cleared, the data-relation objects are automatically deleted.

See also

ADD-PARENT-ID-RELATION( ) method, DEFINE DATASET statement

Data-source object handle

A handle to a data-source object. A data-source object supports the automatic filling of a ProDataSet object member table, as well as applying updates back to one or more database tables. There is a distinct data-source object for each member buffer, which allows a single ProDataSet object and a single FILL operation on that object to combine data from multiple databases. A data-source object is defined independently of any ProDataSet object.

A data-source object can be static or dynamic. A static data-source object is one you define at compile time with the DEFINE DATA-SOURCE statement. A dynamic data-source object is one you create at run time with the CREATE DATA-SOURCE statement. Use this handle to access the data-source object's attributes and methods.

Note: Does not apply to SpeedScript programming.

Syntax

```
data-source-handle [ :attribute | :method ]
```

data-source-handle

A handle variable that references a data-source object.

attribute

An attribute of the data-source object.

method

A method of the data-source object.

Attributes

ADM-DATA attribute	FILL-WHERE-STRING attribute	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	KEYS attribute	MERGE-BY-FIELD attribute
NAME attribute	NEXT-ROWID attribute	NEXT-SIBLING attribute
NUM-SOURCE-BUFFERS attribute	PREFER-DATASET attribute	PRIVATE-DATA attribute
QUERY attribute	RESTART-ROW attribute	RESTART-ROWID attribute
SAVE-WHERE-STRING attribute	TYPE attribute	–

Methods

ADD-SOURCE-BUFFER( ) method	GET-DATASET-BUFFER( ) method
GET-SOURCE-BUFFER( ) method	–

Notes

- To associate a query with a dynamic data-source object, use the QUERY attribute. To disassociate the query and data-source object, set the QUERY attribute to the Unknown value (?).
- To override the WHERE clause in the query, use the FILL-WHERE-STRING attribute.

See also

CREATE DATA-SOURCE statement, DEFINE DATA-SOURCE statement

DEBUGGER system handle

A handle that lets ABL procedures initialize and control the Application Debugger.

To use the DEBUGGER handle, you must have the Application Debugger installed in your OpenEdge environment.

Note: Does not apply to SpeedScript programming.

Syntax

```
DEBUGGER [ :attribute | :method]
```

attribute

Specifies an attribute of the DEBUGGER handle.

method

Specifies a method of the DEBUGGER handle.

Attributes

HANDLE attribute	INSTANTIATING-PROCEDURE attribute	TYPE attribute
VISIBLE attribute	–	–

Methods

CANCEL-BREAK( ) method	CLEAR( ) method (DEBUGGER Handle)
DEBUG( ) method	DISPLAY-MESSAGE( ) method
INITIATE( ) method	SET-BREAK( ) method

Example

The following example displays Orders for each Customer in the `Sports2020` database using two procedure files. The `r-cusbug.p` file initializes the Debugger and sets a breakpoint at line 6 of the `r-ordbug.p` file. Thus, each time `r-ordbug.p` displays an Order, the Debugger takes control before it displays the Order Lines. Just before completing execution, `r-cusbug.p` clears the debugging session before returning.

r-cusbug.p

```
DEFINE NEW SHARED BUFFER CustBuf FOR Customer.

DEFINE VARIABLE debug AS LOGICAL NO-UNDO.

debug = DEBUGGER:INITIATE().
debug = DEBUGGER:SET-BREAK("r-ordbug.p",6).

FOR EACH CustBuf NO-LOCK:
  IF CAN-FIND(Order OF CustBuf) THEN
    RUN r-ordbug.p.
END.

debug = DEBUGGER:CLEAR().
```

r-ordbug.p

```
DEFINE SHARED BUFFER CustBuf FOR Customer.

FOR EACH Order OF CustBuf NO-LOCK:
  DISPLAY CustBuf.Name CustBuf.CustNum CustBuf.City CustBuf.State
    CustBuf.PostalCode Order.OrderNum.
  FOR EACH OrderLine OF Order NO-LOCK, Item OF OrderLine NO-LOCK:
    DISPLAY Item.ItemName Item.ItemLum LrderLine.Qty.
  END. /* FOR EACH OrderLine */
END. /* FOR EACH Order */
```

Notes

- You must initialize the Debugger using either the DEBUG( ) or INITIATE( ) method before using any of the remaining methods in a procedure.

  The DEBUG( ) and INITIATE( ) methods provide separate means to invoke the Debugger, and do not depend on each other to start a debugging session. The DEBUG( ) method initializes and gives control to the Debugger whether or not the INITIATE( ) method has been executed.

- The TYPE attribute returns the widget type, PSEUDO-WIDGET.
- The VISIBLE attribute specifies whether the Debugger window is visible on the screen. When set to FALSE, if the Debugger window is currently visible, it is removed from the screen. When set to TRUE, if the Debugger window is currently invisible, it is displayed. Note that making the Debugger window visible does not, in itself, give control to the Debugger.

  Note: The ABL code that initiates the Debugger and displays it on the screen is responsible for removing the Debugger from the screen when it is no longer needed by setting the VISIBLE attribute to FALSE.

- After invoking the INITIATE method, execution continues in the procedure until it encounters a breakpoint or a statement invoking the DEBUG method. If the procedure encounters a breakpoint, the Debugger takes control running in application mode (with control over the invoking application). If the procedure invokes the DEBUG method, the Debugger takes control running in stand-alone mode (with control only over applications started from the Debugger).
- References to line numbers in internal procedures must be relative to the debug listing in which they are contained.
- When you set or cancel a breakpoint, you must distinguish between a line number value less than 1 and a value of 1 or greater. Any value for *line-number* less than 1 (for example, 0 or -1) specifies the first executable line of the **main procedure** in the file specified by *procedure*. However, a positive value for *line-number* specifies the first executable line on or after *line-number* in the **file** specified by *procedure*. For example, suppose *procedure* specifies a file like this:

```
1       DEFINE VARIABLE lStart AS LOGICAL NO-UNDO INITIAL TRUE.
2       PROCEDURE ShowStart:
3         IF lStart THEN MESSAGE "Procedure is starting ...".
4       END.
5
6       MESSAGE "Hello World!".
7       RUN ShowStart.
8       lStart = FALSE.
          . . .
```

  If you specify a breakpoint at line 0, -1, or any negative value, the breakpoint actually occurs at line 6, the first line that executes in the main procedure. If you specify a breakpoint at line 1 or 2, the breakpoint occurs at line 3, the first executable line in the file, which happens to be the first executable line of an internal procedure.

This distinction also affects procedures containing the Trigger phrase used to define triggers in widget definitions. For example, suppose *procedure* specifies this file:

```
1       DEFINE BUTTON bOK LABEL "OK"
2         TRIGGERS:
3           ON CHOOSE
4             MESSAGE "OK Pressed!".
5         END TRIGGERS.
6       MESSAGE "Hello World!".
          . . .
```

Again, if you specify a breakpoint at line -1, the breakpoint occurs on line 6, but if you specify the breakpoint at line 1, it actually occurs at line 4, which is the first executable line of a trigger block.

Note: You cannot set a watchpoint programmatically using the DEBUGGER system handle. A watchpoint is a form of breakpoint which tells the Debugger to interrupt program execution when the value of a variable, buffer field, or attribute reference changes.

For more information on the Debugger, its features and functions, and its modes of execution, see *Troubleshoot ABL Applications*.

See also

LOG-MANAGER system handle

DEFAULT-WINDOW system handle

A handle to the static window of the current ABL session. Every ABL session has one static window. This window displays all frames until the CURRENT-WINDOW system handle is set to the handle of another [dynamic] window or unless a displayed frame is explicitly parented to another window. Its message area also displays messages that have no other available display destination. Use the DEFAULT-WINDOW handle to set or examine the attributes of this unnamed session window.

Syntax

```
DEFAULT-WINDOW [ :attribute ]
```

attribute

An attribute of the DEFAULT-WINDOW.

Attributes

The DEFAULT-WINDOW handle has all the attributes of a window widget.

Notes

- If you display or view a frame without the IN WINDOW option and without first explicitly or implicitly setting CURRENT-WINDOW to the handle of a non-default window, the ABL virtual machine (AVM) displays the frame in the default window.

- If the CURRENT-WINDOW does not have a message area and you execute the MESSAGE statement without an explicit or implicit VIEW-AS ALERT-BOX option, the AVM displays the default window in order to display the message in its own message area.
- In a character interface, or at the start of a GUI client session, the ACTIVE-WINDOW, CURRENT-WINDOW, and DEFAULT-WINDOW handles all return the handle of the static window for the current ABL session.
- You can make the default window the current window by assigning DEFAULT-WINDOW to CURRENT-WINDOW.
- You can enable or disable the default window by changing the value of DEFAULT-WINDOW:SENSITIVE (the SENSITIVE attribute).
- You can set the menu bar for the default window by assigning the handle of a menu bar to DEFAULT-WINDOW:MENUBAR (the MENUBAR attribute).
- You can make the default window visible or invisible by changing the value of DEFAULT-WINDOW:VISIBLE (the VISIBLE attribute).
- Because DEFAULT-WINDOW is the initial default parent for ABL frames, you cannot set DEFAULT-WINDOW to the handle of a .NET form's shadow window. If allowed, this would make the associated .NET form a possible parent for ABL frames, and ABL cannot parent ABL frames to .NET forms. A .NET form's shadow window handle is provided by the ProWinHandle property of the Progress.Windows.Form class, which an OpenEdge .NET class designed for instantiating .NET forms in an ABL session. Therefore, any attempt to set DEFAULT-WINDOW to a ProWinHandle property value raises a run-time error.

See also

ACTIVE-WINDOW system handle, CURRENT-WINDOW system handle

DSLOG-MANAGER system handle

A handle to logging settings for the DataServers in the current session. The DataServers use a separate log file which you can specify with the DataServer Logging (`-dslog`) startup parameter. The default filename is `dataserv.lg`.

Note: This system handle applies to the DataServer for Microsoft SQL Server as well as the DataServer for Oracle.

Syntax

```
DSLOG-MANAGER [ :attribute | :method ]
```

attribute

An attribute of the DSLOG-MANAGER system handle.

method

A method of the DSLOG-MANAGER system handle.

Attributes

DB-CONTEXT attribute	ENTRY-TYPES-LIST attribute	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	LOG-ENTRY-TYPES attribute	LOG-THRESHOLD attribute
LOGFILE-NAME attribute	LOGGING-LEVEL attribute	NUM-LOG-FILES attribute
TYPE attribute	–	–

The following table lists the DSLOG-MANAGER system handle attributes.

Table 65: DSLOG-MANAGER system handle attributes

Attribute	Access	Function
DB-CONTEXT[76]	Readable/ Writable	Specifies the logical database name of a valid and connected data source for a supported db-type. It sets the system handle to a specific server context to enable access to other DSLOG-MANAGER attributes and methods. You can also set it to the alias of a connected data source.
ENTRY-TYPES-LIST	Readable	Returns a list of all valid entry types.
LOG-ENTRY-TYPES	Readable/ Writable	Returns/Writes a list of one or more types of log entries. The default value returned by the LOG-ENTRY-TYPES attribute are the logentrytypes specified in the -Dsrv parameter when the foreign data source is connected.
LOGFILE-NAME[77]	Readable	Returns the name of the log file. By default, the DSLOG-MANAGER uses the dataserv.lg log file. You can change the file name by using the -dslog startup parameter from the OpenEdge client. The server context cannot work without a log file opened so it cannot be changed at runtime. This would risk having no log file opened in the event that the new log file name failed to open.
LOGGING-LEVEL	Readable/ Writable	Returns/Writes the level at which log entries are written. The default value returned by the LOGGING-LEVEL attribute is the logginglevel specified in the -Dsrv parameter when the foreign data source is connected.

[76] Each connection to a DataServer is its own entity. The DB-CONTEXT attribute enables you to provide different logging settings for each data source connection. The DB-CONTEXT attribute must be set to the logical database name or alias which is currently connected, prior to setting any other attributes. The attribute defaults to the unknown value. If DB-CONTEXT does not contain the value of a valid logical database name, any attempt to access other attributes of DSLOG-MANAGER fails.

[77] The value returned by the LOGFILE-NAME is the default name dataserv.lg or the name specified by the -dslog client startup parameter for self-service clients or the name specified by the -dslog startup parameter when ProBroker is started.

Attribute	Access	Function
LOG-THRESHOLD[78]	Readable	Returns the file size threshold of log files. When the current log file becomes equal to or greater than the specified size, OpenEdge renames and saves the log file and creates a new log file.
NUM-LOG-FILES[79]	Readable	Returns the number of rolled over log files to keep on disk at any one time, for OpenEdge session, including the current log file.
HANDLE	Readable	Returns a handle.
INSTANTIATING-PROCEDURE	Readable	Returns a procedure handle for the first, or instantiating, procedure.
TYPE	Readable	Returns the type of handle.

Methods

CLEAR-LOG( ) method	WRITE-MESSAGE( ) method

For more information about DataServer logging, see *Troubleshoot ABL Applications*, *Use the Microsoft SQL Data Server*, and *Use the Oracle Data Server*.

ERROR-STATUS system handle

A handle to error information on the last statement executed with the NO-ERROR option.

Syntax

```
ERROR-STATUS [ :attribute | :method ]
```

attribute

Specifies an attribute of the ERROR-STATUS handle.

method

Specifies a method of the ERROR-STATUS handle.

[78] The value returned by the LOG-THRESHOLD is the name specified by the -logthreshold client startup parameter for self-service clients or the name specified by the -logthreshold startup parameter when ProBroker is started.

[79] The value returned by the NUM-LOG-FILES is the name specified by the -numlogfiles client startup parameter for self-service clients or the value specified by the -numlogfiles startup parameter when ProBroker is started.

Attributes

ERROR attribute	ERROR-OBJECT-DETAIL attribute	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	NUM-MESSAGES attribute	TYPE attribute

Methods

GET-MESSAGE( ) method (Handle)	GET-NUMBER( ) method

Examples

The following example uses the NO-ERROR and the ERROR-STATUS handle extensively to demonstrate when ERROR-STATUS attributes are reset:

r-errst1.p

```
CONNECT "db-xyz" NO-ERROR.
RUN chk-connect NO-ERROR.
IF ERROR-STATUS:ERROR THEN
  MESSAGE "Run statement failed.".

PROCEDURE chk-connect:
  DEFINE VARIABLE connect-ok AS LOGICAL NO-UNDO INITIAL TRUE.

  IF ERROR-STATUS:ERROR THEN DO:
    MESSAGE "Connect failed.".
    connect-ok = FALSE NO-ERROR.
    IF ERROR-STATUS:ERROR THEN
      MESSAGE "Assignment failed.".
  END.

  IF connect-ok THEN RETURN "OK".
   ELSE RETURN "FAILED".
END PROCEDURE.
```

Within the internal procedure, chk-connect, the first reference to ERROR-STATUS:ERROR returns status on the CONNECT statement from the main procedure. The second reference returns status on the assignment statement. The reference to ERROR-STATUS:ERROR in the main procedure returns status on the RUN statement. Note that the ERROR-STATUS attributes are set only after the statement with NO-ERROR completes. Therefore the references in the internal procedure are not affected by the RUN statement itself.

The following procedure accepts a character string value and lets you convert it to one of several data types. The internal convert procedure attempts the conversion. If the conversion is successful, it displays the converted value. If the conversion is unsuccessful, the ERROR-STATUS handle holds error information. After running convert, the CHOOSE trigger checks ERROR-STATUS:ERROR and ERROR-STATUS:NUM-MESSAGES to determine if either error information is available or messages have been returned, even if ERROR is not raised. If either condition is true, this lets you view this information. ABL includes many errors that generate messages, but do not raise the ERROR condition, such as most errors generated by ABL built-in functions and handle methods.

r-errsts.p

```
DEFINE VARIABLE txt AS CHARACTER NO-UNDO FORMAT "X(20)".
DEFINE VARIABLE ix  AS INTEGER   NO-UNDO.
```

```
DEFINE BUTTON b_int  LABEL "Integer".
DEFINE BUTTON b_date LABEL "Date".
DEFINE BUTTON b_dec  LABEL "Decimal".
DEFINE BUTTON b_log  LABEL "Logical".
DEFINE BUTTON b_quit LABEL "Quit" AUTO-ENDKEY.

DEFINE FRAME butt-frame
  b_int b_date b_dec b_log b_quit
  WITH CENTERED ROW SCREEN-LINES - 2.
DEFINE FRAME get-info
  txt LABEL "Enter Data To Convert"
  WITH ROW 2 CENTERED SIDE-LABELS TITLE "Data Conversion - Error Check".

ON CHOOSE OF b_int, b_date, b_dec, b_log IN FRAME butt-frame DO:
  IF txt:MODIFIED IN FRAME get-info THEN DO:
    ASSIGN txt.
    RUN convert(txt).
    IF ERROR-STATUS:ERROR OR ERROR-STATUS:NUM-MESSAGES > 0 THEN DO:
      MESSAGE ERROR-STATUS:NUM-MESSAGES
        " errors occurred during conversion." SKIP
        "Do you want to view them?"
        VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO
        UPDATE view-errs AS LOGICAL.
      IF view-errs THEN
      DO ix = 1 TO ERROR-STATUS:NUM-MESSAGES:
        MESSAGE ERROR-STATUS:GET-NUMBER(ix) ERROR-STATUS:GET-MESSAGE(ix).
      END.
    END.
  END. /* IF txt:MODIFIED... */
  ELSE
    MESSAGE "Please enter data to be converted, then choose the type of
conversion to perform."
      VIEW-AS ALERT-BOX MESSAGE BUTTONS OK.
END.

ENABLE ALL WITH FRAME butt-frame.
ENABLE txt WITH FRAME get-info.
WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame FOCUS txt IN FRAME get-info.

PROCEDURE convert:
  DEFINE INPUT PARAMETER cText AS CHARACTER NO-UNDO.

  DEFINE VARIABLE dValue AS DATE    NO-UNDO.
  DEFINE VARIABLE fValue AS DECIMAL NO-UNDO.
  DEFINE VARIABLE iValue AS INTEGER NO-UNDO.
  DEFINE VARIABLE lValue AS LOGICAL NO-UNDO.

  MESSAGE SELF:LABEL.

  CASE SELF:LABEL:
    WHEN "Integer" THEN DO:
      ASSIGN iValue = INTEGER(cText) NO-ERROR.
      MESSAGE "Converted value:" iValue.
    END.
    WHEN "Date" THEN DO:
      ASSIGN dValue = DATE(INTEGER(SUBSTRING(cText,1,2)),
                       INTEGER(SUBSTRING(cText,4,2)),
                       INTEGER(SUBSTRING(cText,7)) ) NO-ERROR.
      MESSAGE "Converted value:" dValue.
    END.
    WHEN "Decimal" THEN DO:
      ASSIGN fValue = DECIMAL(cText) NO-ERROR.
      MESSAGE "Converted value:" fValue.
    END.
    WHEN "Logical" THEN DO:
      ASSIGN lValue = (cText = "yes" OR cText = "true") NO-ERROR.
      MESSAGE "Converted value:" lValue.
    END.
```

```
  END.
END PROCEDURE.
```

Notes

- The ERROR attribute indicates whether the ERROR condition was raised during the execution of the last statement that contained the NO-ERROR option. Some errors may occur without raising the ERROR condition. For example, compiler errors do not raise the ERROR condition.
- The NUM-MESSAGES attribute indicates the total number of errors that occurred during that statement.
- The ERROR-OBJECT-DETAIL attribute identifies a SOAP-fault object that contains SOAP fault message detail.

  If a Web service operation generates a SOAP fault message, the AVM generates the following error:

  ```
  Web service %s<operation> failed.  SOAP faultstring is %s (nnnn)
  ```

  The complete SOAP fault error message is returned to the AVM as part of the ERROR-STATUS system handle.

  If the ABL application invokes the Web service operation with the NO-ERROR option on the RUN statement, any errors that occur as a result of the operation are suppressed. In this case, the application can access the SOAP fault message detail using the SOAP-fault and SOAP-fault-detail object handles. Otherwise, the AVM displays the error message to the end user.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.
- The GET-MESSAGE method and the GET-NUMBER method let you access the error numbers and messages for all errors that occurred during the execution of the last statement with the NO-ERROR option.
- Usually, the NO-ERROR option on a statement suppresses the display of error messages. However, if a STOP condition occurs, the error message is written to the windows. These messages are also available through the ERROR-STATUS attributes. For example, the STOP condition is raised when a procedure to be run is not found. Two specific instances of this are:
  - If you use NO-ERROR on a RUN statement and the procedure is not found or cannot compile.
  - If you execute a data handling statement, such as DELETE with the NO-ERROR option and the corresponding trigger procedure is not found or cannot compile.

See also

SOAP-fault object handle, SOAP-fault object handle

FILE-INFO system handle

A handle to an operating system file.

Syntax

```
FILE-INFO [ :attribute ]
```

attribute

Specifies an attribute of the FILE-INFO handle.

Attributes

FILE-CREATE-DATE attribute	FILE-CREATE-TIME attribute	FILE-MOD-DATE attribute
FILE-MOD-TIME attribute	FILE-NAME attribute	FILE-SIZE attribute
FILE-TYPE attribute	FULL-PATHNAME attribute	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	PATHNAME attribute	TYPE attribute

Example

After you set the value of the FILE-NAME attribute, you can read the values of the other attributes. For example:

r-osfile.p

```
DEFINE VARIABLE os-file AS CHARACTER NO-UNDO FORMAT "x(60)" LABEL "File".

REPEAT:
  SET os-file WITH FRAME osfile-info.
  FILE-INFO:FILE-NAME = os-file.
  DISPLAY FILE-INFO:FULL-PATHNAME FORMAT "x(60)" LABEL "Full Path"
    FILE-INFO:PATHNAME FORMAT "x(60)" LABEL "Path"
    FILE-INFO:FILE-TYPE LABEL "Type"
    WITH FRAME osfile-info SIDE-LABELS TITLE "OS File Info".
END.
```

Notes

- The FILE-NAME, FULL-PATHNAME, and PATHNAME attributes can hold names that contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.
- You cannot use the FILE-INFO handle to by-pass operating system security. You must have read access to the file and the directory that contains it to obtain information through FILE-INFO.
- These attributes return the Unknown value (?) until they are set, and also if the specified file cannot be found or you do not have permission to access the file.
- If you set the FILE-NAME attribute to a relative pathname, the FILE-INFO handle searches the current PROPATH to locate the file.
- The FILE-TYPE attribute returns a string containing exactly one of the following file type characters:
  - **D** — If the file is a directory
  - **F** — If the file is a standard file or FIFO pipe (UNIX systems)
  - **M** — If the file is a member of an ABL procedure library

- **S** — If the file is a special device (UNIX systems)
- **X** — If the file type is unknown (contact your Progress Software Corporation Technical Support representative when you receive this value)

The attribute string can contain any of the following file type characters:

- **H** — If the file is hidden
- **L** — If the file is a symbolic link (UNIX systems)
- **P** — If the file is a pipe file (UNIX systems)
- **R** — If the file is readable
- **W** — If the file is writable

- The FULL-PATHNAME attribute returns the absolute pathname of the file specified in the FILE-NAME attribute.
- If the FILE-NAME attribute contains a simple filename or relative pathname, the PATHNAME attribute contains the pathname of the specified file starting with the directory on the PROPATH where it is found. Otherwise, the PATHNAME attribute contains the absolute pathname specified in the FILE-NAME attribute.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.

FOCUS system handle

A handle to the field-level widget that is the current field.

Note: Does not apply to SpeedScript programming.

Syntax

```
FOCUS [ :attribute ]
```

attribute

An attribute of the widget that has current input focus.

Attributes

The specific attributes available depend on the type of the widget. You can determine the widget type by examining the FOCUS:TYPE attribute.

Example

The following example uses the FOCUS handle to provide helpful information to the user. The procedure displays an interface that contains several different types of widgets. If you type ?, the procedure displays a message specifying the type of widget that has focus and whether VALUE-CHANGED event is a valid event for that widget.

r-focus.p

```
DEFINE VARIABLE inv-price   LIKE item.price NO-UNDO.
DEFINE VARIABLE inv-value   LIKE item.price NO-UNDO.
DEFINE VARIABLE report-type AS INTEGER      NO-UNDO INITIAL 1.

DEFINE BUTTON ok-butt     LABEL "OK" AUTO-GO.
DEFINE BUTTON cancel-butt LABEL "CANCEL" AUTO-ENDKEY.

FORM
  inv-price LABEL "Price" AT ROW 1.25 COLUMN 2
  report-type LABEL "Report Sorted ..." AT ROW 2.25 COLUMN 2
    VIEW-AS RADIO-SET
      RADIO-BUTTONS "By Catalog Page", 1, "By Inventory Value", 2 SKIP
  ok-butt cancel-butt
  WITH FRAME select-frame SIDE-LABELS.

ON ? ANYWHERE DO:
  MESSAGE "This is a" FOCUS:TYPE + ". VALUE-CHANGED is"
    (IF VALID-EVENT(FOCUS, "VALUE-CHANGED") THEN "a" ELSE "NOT a")
    "valid event for this widget."
    VIEW-AS ALERT-BOX INFORMATION BUTTONS OK.
  RETURN NO-APPLY.
END.

ENABLE ALL WITH FRAME select-frame.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Note that this example prevents you from entering the question mark character (?) in any field. This does not cause a problem in `r-focus.p` because a question mark is not a valid input character for any field in the interface.

Notes

- A typical use of the FOCUS handle identifies the widget that contains the current text selection for reference by the system clipboard. For an example of this usage, see *OpenEdge Programming Interfaces*.
- Within a WAIT-FOR statement, you can specify the field that receives initial input focus.
- You must give input focus to any fill-in widget where you want to set the AUTO-ZAP attribute. For more information, see the SAX-reader object handle reference entry.

See also

SELF system handle, WAIT-FOR statement (ABL only)

FONT-TABLE system handle
(Windows only; Graphical interfaces only)

A handle to the current font table.

Note: Does not apply to SpeedScript programming.

Syntax

```
FONT-TABLE [ :attribute | :method ]
```

attribute

Specifies an attribute of the FONT-TABLE handle.

method

Specifies a method of the FONT-TABLE handle.

Attributes

HANDLE attribute	INSTANTIATING-PROCEDURE attribute	NUM-ENTRIES attribute
TYPE attribute	–	–

Methods

GET-TEXT-HEIGHT-CHARS( ) method	GET-TEXT-HEIGHT-PIXELS( ) method
GET-TEXT-WIDTH-CHARS( ) method	GET-TEXT-WIDTH-PIXELS( ) method

Example

This code shows how to query and set the integer attribute, NUM-ENTRIES:

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

ix = FONT-TABLE:NUM-ENTRIES. /* to query */
/* or */
ix = 255.
FONT-TABLE:NUM-ENTRIES = ix. /* to set */
```

Notes

- Unlike the COLOR-TABLE system handle, the FONT-TABLE system handle does not allow you to set fonts dynamically. Font entries can only be changed by the user through the font system dialog box. Fonts are always dynamic.
- The current font table is the font table in the current environment, which is the startup environment or the environment most recently specified in a USE statement.
- To determine the number of font entries in the font table, query the NUM-ENTRIES attribute.
- To change the number of font entries in the font table, set the NUM-ENTRIES attribute.
- To allow users to set dynamic font table entries at run time, an application can display a font common dialog with the SYSTEM-DIALOG FONT statement.

- To save font definitions from the font table to the current environment file, use the PUT-KEY-VALUE statement. To retrieve the font definition specified in the current environment file, use the GET-KEY-VALUE statement.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.

See also

GET-KEY-VALUE statement, PUT-KEY-VALUE statement, SYSTEM-DIALOG FONT statement, USE statement

---

LAST-EVENT system handle

A handle to the last event the application received.

Syntax

```
LAST-EVENT [ :attribute ]
```

attribute

An attribute of the LAST-EVENT.

Attributes

CODE attribute	COLUMN attribute	EVENT-TYPE attribute
FUNCTION attribute	HANDLE attribute	INSTANTIATING-PROCEDURE attribute
LABEL attribute	ON-FRAME-BORDER attribute	ROW attribute
TYPE attribute	WIDGET-ENTER attribute	WIDGET-LEAVE attribute
X attribute	Y attribute	–

Example

This procedure creates a variety of widgets and a frame that acts as a message area. As you move around the widgets the procedure tells you what events ABL generates.

r-lstevt.p

```
DEFINE VARIABLE msg_watcher  AS CHARACTER NO-UNDO FORMAT "x(50)".
DEFINE VARIABLE fi           AS CHARACTER NO-UNDO FORMAT "x(50)"
  LABEL "Fill-in".
DEFINE VARIABLE edit         AS CHARACTER NO-UNDO
  VIEW-AS EDITOR SIZE 10 BY 2.
DEFINE VARIABLE txt0         AS CHARACTER NO-UNDO FORMAT "x(75)".

DEFINE BUTTON exit LABEL "Go to Exit" AUTO-GO.
DEFINE BUTTON test LABEL "Test" SIZE 15 BY 3.

DISPLAY "Feel free to move around..." VIEW-AS TEXT SKIP
  fi SKIP(0.1)
  edit AT 2 SKIP(0.1)
  test AT 10 exit AT 50 SKIP(0.5)
  WITH FRAME f SIDE-LABELS.

DISPLAY txt0 LABEL "LAST-EVENT:LABEL FUNCTION (TYPE) > SELF:TYPE LABEL"
  WITH FRAME report 4 DOWN CENTERED TITLE "Messages".

ON RETURN, TAB, ANY-PRINTABLE, GO ANYWHERE
  RUN msgwatch. /* This procedure uses LAST-EVENT. */

ENABLE ALL WITH FRAME f.
WAIT-FOR GO OF FRAME f FOCUS fi.

PROCEDURE msgwatch.
  txt0 = LAST-EVENT:LABEL + " " + LAST-EVENT:FUNCTION + " [" +
         LAST-EVENT:EVENT-TYPE + "]-->" + SELF:TYPE + " "
  txt0 = txt0 + (IF CAN-QUERY(SELF, "LABEL") THEN
           SELF:LABEL ELSE STRING(SELF).

  DISPLAY txt0 WITH FRAME report.
  DOWN WITH FRAME report.
END PROCEDURE.
```

Notes

- For keyboard events, the CODE, FUNCTION, and LABEL attributes return the key code, key function, and key label of the event, respectively. For all other events the CODE attribute returns the numeric event code.

  For mouse events, the FUNCTION attribute returns the names of portable mouse events and the LABEL attribute returns the names of three-button mouse events.

  For high-level ABL events, the FUNCTION attribute returns the name of the event. If the ABL event is triggered by a key press, the LABEL attribute returns the key label. Otherwise, it returns the event name, as with the FUNCTION attribute.

- The EVENT-TYPE attribute returns the category of the event: KEYPRESS, MOUSE, or ABL.
- The ON-FRAME-BORDER attribute indicates whether a MOUSE event occurred in the border of a frame.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.
- The X and Y attributes return the pixel position of a MOUSE event relative to the current frame.
- For browse widgets, WIDGET-ENTER and WIDGET-LEAVE are different depending on whether the browse is editable or read-only. For editable browse widgets, WIDGET-ENTER contains the handle of the column with focus. For read-only browse widgets, WIDGET-ENTER contains the handle of the browse. For editable brows widgets, WIDGET-LEAVE contains the handle of the column the user just left. For read-only browse widgets, WIDGET-LEAVE contains the handle of the field-level widget the user just left.

See also

LIST-EVENTS function, LIST-WIDGETS function, SELF system handle, VALID-EVENT function

LOG-MANAGER system handle

A handle to logging settings for the current ABL session.

Syntax

```
LOG-MANAGER [ :attribute | :method ]
```

attribute

An attribute of the LOG-MANAGER system handle.

method

A method of the LOG-MANAGER system handle.

Attributes

ENTRY-TYPES-LIST attribute	HANDLE attribute	INSTANTIATING-PROCEDURE attribute
LOG-ENTRY-TYPES attribute	LOGFILE-NAME attribute	LOGGING-LEVEL attribute
LOG-THRESHOLD attribute	NUM-LOG-FILES attribute	TYPE attribute

Methods

CLEAR-LOG( ) method	CLOSE-LOG( ) method
WRITE-MESSAGE( ) method	–

For more information about logging, see *Troubleshoot ABL Applications*.

Procedure object handle

A handle that can reference the context of one of the following procedure object instances:

- A persistent procedure in the current session
- A persistent procedure in a remote session (on an application server)
- A single-run or singleton procedure in a remote session (on an application server)
- A running procedure in the current session, including the currently executing procedure or any other procedure currently on the session call stack

- A procedure object that encapsulates a Web service operation (*Web service procedure object*)

This object allows you to read and modify the context of the specified procedure according to the type of procedure object instance.

Syntax

```
procedure-handle [ :attribute | :method ]
```

procedure-handle

A handle variable that references a procedure object.

For Web services, this object is instantiated when you execute the RUN ON statement that references a Web service server object.

attribute

An attribute of a procedure handle.

method

Specifies a method of a procedure handle.

Attributes

ADM-DATA attribute	ASYNC-REQUEST-COUNT attribute	CURRENT-WINDOW attribute
DB-REFERENCES attribute	FILE-NAME attribute	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	INTERNAL-ENTRIES attribute	NAME attribute
NEXT-SIBLING attribute	PERSISTENT attribute	PREV-SIBLING attribute
PRIVATE-DATA attribute	PROXY attribute	PUBLISHED-EVENTS attribute
REMOTE attribute	SERVER attribute	SUPER-PROCEDURES attribute
SINGLE-RUN attribute	SINGLETON attribute	TRANSACTION attribute
TYPE attribute	UNIQUE-ID attribute	–

Methods

ADD-SUPER-PROCEDURE( ) method	GET-SIGNATURE( ) method
REMOVE-SUPER-PROCEDURE( ) method	SET-CALLBACK( ) method
SET-CALLBACK-PROCEDURE( ) method	–

Example

The following procedure runs another procedure (`r-thispr.p`) persistently, setting a procedure object handle for it, and displays the value of its PRIVATE-DATA attribute as a message:

r-run-thispr-pers.p

```
DEFINE VARIABLE hthispr AS HANDLE NO-UNDO.

RUN r-thispr.p PERSISTENT SET hthispr.

MESSAGE "Running the " hthispr:PRIVATE-DATA VIEW-AS ALERT-BOX INFORMATION.
```

For more information on the `r-thispr.p` example procedure, see the reference entry for the THIS-PROCEDURE system handle in this section.

Notes

- You can also define triggers for procedure object handles.
- To create an instance of a persistent procedure, use the PERSISTENT option of the RUN statement, as shown for the `r-run-thispr-pers.p` example procedure. For other examples of creating persistent procedures, see the reference entry for the RUN statement.
- You can access the handles and attributes of all persistent procedure instances in a session using the FIRST-PROCEDURE or LAST-PROCEDURE attribute of the SESSION handle.

See also

RUN statement, SESSION system handle, THIS-PROCEDURE system handle

---

ProDataSet object handle

A handle to a ProDataSet object. A ProDataSet object is a collection of one or more related temp-tables. Each temp-table in a ProDataSet object can attach to a data-source object that allows filling of the temp-table from the data source, or updating the data source from the temp-table. A ProDataSet object can optionally contain a set of data relations between the temp-tables.

A ProDataSet object can be static or dynamic. A static ProDataSet object is one you define at compile time with the DEFINE DATASET statement. A dynamic ProDataSet object is one you create at run time with the CREATE DATASET statement. Use this handle to access the ProDataSet object's attributes and methods, and its sub-elements (its temp-table buffers, data relations, data sources, and so on) and their attributes and methods.

Syntax

```
dataset-object-handle [ :attribute | :method ]
```

dataset-object-handle

An item of type HANDLE representing a handle to a ProDataSet object.

attribute

An attribute of the ProDataSet object.

method

A method of the ProDataSet object.

Attributes

ADM-DATA attribute	DATA-SOURCE-MODIFIED attribute	DYNAMIC attribute
ERROR attribute	HANDLE attribute	INSTANTIATING-PROCEDURE attribute
NAME attribute	NAMESPACE-PREFIX attribute	NAMESPACE-URI attribute
NEXT-SIBLING attribute	NUM-BUFFERS attribute	NUM-REFERENCES attribute
NUM-RELATIONS attribute	NUM-TOP-BUFFERS attribute	PRIVATE-DATA attribute
REJECTED attribute	RELATIONS-ACTIVE attribute	SERIALIZE-NAME attribute
TOP-NAV-QUERY attribute	TYPE attribute	UNIQUE-ID attribute
XML-NODE-NAME attribute	XML-NODE-TYPE attribute	–

Methods

ACCEPT-CHANGES( ) method	ADD-BUFFER( ) method
ADD-PARENT-ID-RELATION( ) method	ADD-RELATION( ) method
APPLY-CALLBACK( ) method	CLEAR( ) method (ProDataSet Object Handle)
COPY-DATASET( ) method	COPY-TEMP-TABLE( ) method
CREATE-LIKE( ) method	EMPTY-DATASET( ) method
FILL( ) method	GET-BUFFER-HANDLE( ) method
GET-CALLBACK-PROC-CONTEXT( ) method	GET-CALLBACK-PROC-NAME( ) method
GET-CHANGES( ) method	GET-RELATION( ) method
GET-TOP-BUFFER( ) method	MERGE-CHANGES( ) method
READ-JSON( ) method	READ-XML( ) method
READ-XMLSCHEMA( ) method	REJECT-CHANGES( ) method
SET-BUFFERS( ) method	SET-CALLBACK( ) method

SET-CALLBACK-PROCEDURE( ) method	WRITE-JSON( ) method
WRITE-XML( ) method	WRITE-XMLSCHEMA( ) method

Events

AFTER-FILL event	BEFORE-FILL event

For information on these FILL events, see the ProDataSet events.

Notes

For information about dynamically accessing the data in a ProDataSet object, see the reference entries related to the TEMP-TABLE, BUFFER, BUFFER-FIELD, and QUERY objects.

See also

Buffer object handle, CREATE DATASET statement, DEFINE DATASET statement, EMPTY-TEMP-TABLE( ) method, Temp-table object handle

PROFILER system handle

A handle to initialize and control the profiler. The output from the profiler provides processing time and call-tree information to locate bottlenecks. Developers can quickly identify where an application is spending the most processing time and navigate through the call tree to the potential bottlenecks.

Syntax

```
PROFILER [ :attribute | :method]
```

attribute

Specifies an attribute of the PROFILER handle.

method

Specifies a method of the PROFILER handle.

Attributes

COVERAGE attribute	DESCRIPTION attribute	DIRECTORY attribute
ENABLED attribute	FILE-NAME attribute	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	LISTINGS attribute	PROFILING attribute

STATISTICS attribute	TRACE-FILTER attribute	TRACING attribute
TYPE attribute		

Methods

USER-DATA ( ) method	WRITE-DATA( ) method

Notes

The best practice for profiling applications during development is to use the **Run** > **Run Configurations** in Progress Developer Studio to enable and set the profiler properties on the **Profiler** tab. This automatically sets the profiler attributes and writes the output to a file which automatically opens in the Profiler view. For more information, see Profiler View.

Developers who want to instrument the profiler using the PROFILER system handle must set mandatory attributes, optional attributes, enable profiling, and write the profiler data to the designated output file. The order in which the attributes are set matters.

1. Set the ENABLED attribute.

   Note: Enabling and disabling profiling triggers the following actions on the profiler data output file, as listed below:

   - If you set PROFILER:ENABLED = FALSE then profiling is disabled and the AVM writes the accumulated profiling data to the output file, and discards the registry of entries containing the list of active procedures and their module identifiers.
   - Profiling is automatically disabled if the ABL session exits while PROFILER:ENABLED = TRUE or if the AVM shutsdown. This ensures that the profiling information is written to the output file.

2. Set the FILE-NAME attribute.

   Note: If you do not want the profiler information in the file to be overwritten by a subsequent call to WRITE-DATA() then ensure that you provide a new FILE-NAME for the next profiling session and subsequent invocation of WRITE-DATA(). WRITE-DATA() can be called explicitly or by setting PROFILER:ENABLED = FALSE.

3. Set the DESCRIPTION attribute
4. Set the optional attributes: COVERAGE attribute, DIRECTORY attribute, LISTINGS attribute, STATISTICS attribute TRACE-FILTER attribute and TRACING attribute.
5. Turn on profiling with the PROFILING attribute.
6. Run the application you want to profile.
7. You can write the profiling information to an output file in one of two ways:
   - Disable profiling by setting PROFILER:ENABLED = FALSE to automatically write the profiler data to the output file.
   - Use the WRITE-DATA( ) method.

Note: If you wish to capture subsequent profiler data, then ensure that you change the file name using PROFILER:FILE-NAME attribute. If you don't provide a new file name then the previous file with the profiler data is overwritten.

8. Turn off profiling when no longer needed with the PROFILING attribute.

The output file can be opened in the Profiler view in Progress Developer Studio.

See also

Profiler View, Profiler (-profile), Server-side profiling

Query object handle

A handle to a query object. A query object corresponds to an underlying ABL query, which can be static or dynamic. An example of a static underlying query is one you define at compile time with the DEFINE QUERY statement. An example of a dynamic underlying query is one you create at run time with the new CREATE QUERY statement.

Syntax

```
query-handle [ :attribute | :method ]
```

query-handle

An item of type HANDLE representing a handle to a query object.

attribute

An attribute of the query object.

method

A method of the query object.

Attributes

ADM-DATA attribute	BASIC-LOGGING attribute	CACHE attribute
CURRENT-RESULT-ROW attribute	DYNAMIC attribute	FORWARD-ONLY attribute
HANDLE attribute	INDEX-INFORMATION attribute	INSTANTIATING-PROCEDURE attribute
IS-OPEN attribute	NAME attribute	NUM-BUFFERS attribute
NUM-RESULTS attribute	PREPARE-STRING attribute	PRIVATE-DATA attribute

QUERY-OFF-END attribute	SKIP-DELETED-RECORD attribute	TYPE attribute
UNIQUE-ID attribute	—	—

Methods

ADD-BUFFER( ) method	APPLY-CALLBACK( ) method
CREATE-RESULT-LIST-ENTRY( ) method	DELETE-RESULT-LIST-ENTRY( ) method
DUMP-LOGGING-NOW( ) method	FIRST-OF( ) method
GET-BUFFER-HANDLE( ) method	GET-CALLBACK-PROC-CONTEXT( ) method
GET-CALLBACK-PROC-NAME( ) method	GET-CURRENT( ) method
GET-FIRST( ) method	GET-LAST( ) method
GET-NEXT( ) method	GET-PREV( ) method
LAST-OF( ) method	QUERY-CLOSE( ) method
QUERY-OPEN( ) method	QUERY-PREPARE( ) method
REPOSITION-BACKWARD( ) method	REPOSITION-FORWARD( ) method
REPOSITION-TO-ROW( ) method	REPOSITION-TO-ROWID( ) method
SET-BUFFERS( ) method	SET-CALLBACK( ) method
SET-CALLBACK-PROCEDURE( ) method	–

See also

Buffer object handle, Buffer-field object handle

RCODE-INFO handle

A handle to a specific ABL r-code file.

Syntax

```
RCODE-INFO [ :attribute ]
```

attribute

Specifies an attribute of the RCODE-INFO handle.

Attributes

CODEPAGE attribute	CRC-VALUE attribute	DB-REFERENCES attribute
DISPLAY-TYPE attribute	FILE-NAME attribute	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	IS-CLASS attribute	LABELS-HAVE-COLONS attribute
LANGUAGES attribute	MD5-VALUE attribute	SIGNATURE-VALUE attribute
TABLE-CRC-LIST attribute	TABLE-LIST attribute	TYPE attribute
UNDO-THROW-SCOPE attribute	—	—

Example

The following example prompts for the name of an r-code file and returns its CRC code and the languages for which it is compiled:

r-rcode.p

```
DEFINE VARIABLE rcode-file  AS CHARACTER NO-UNDO FORMAT "x(60)" LABEL "File".

REPEAT:
  SET rcode-file WITH FRAME rc-info.
  RCODE-INFO:FILE-NAME = rcode-file.
  DISPLAY RCODE-INFO:CRC-VALUE LABEL "CRC"
    RCODE-INFO:LANGUAGES FORMAT "x(60)" LABEL "Languages"
    WITH FRAME rc-info SIDE-LABELS TITLE "R-code Check".
END.
```

Notes

- In Windows, the filename cannot contain characters outside of the non-Unicode code page. See *Internationalize ABL Applications* for more information about Unicode and code pages.
- ABL generates an r-code file when you compile a procedure with the SAVE option of the COMPILE statement. You **cannot** use the RCODE-INFO handle to get information on session compiles.
- To use the RCODE-INFO handle, you must first set the FILE-NAME attribute to the name of an r-code file (with or without a `.r` or `.p` extension). If you do not provide a full pathname, ABL searches your `PROPATH` to find the file. You can then read the CRC-VALUE attribute and LANGUAGES attribute to get information on the file. If the r-code file is not found, both LANGUAGES and CRC-VALUE are set to the Unknown value (?).
- The LANGUAGES attribute holds a comma-separated list of language names supported by the r-code. The default segment appears in the list as the value `<unnamed>`.
- The CRC-VALUE attribute returns the r-code CRC value stored in the r-code. The calculation for this value is based on the filename and contents of the procedure file during compilation. This value is different from any database CRCs that are stored in the r-code. For more information on CRCs, see *Manage ABL Applications*.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.
- The DISPLAY-TYPE attribute holds a value of "`GUI`" for a graphical display, "`TTY`" for a character-mode display, or a blank string ("") for code that does not contain any display-specific statements. Use of this

attribute allows developers to determine if the r-code contains statements that make it executable only in the same display environment on which it was compiled.

SAX-attributes object handle

Contains the list of attribute names, values, and optional namespaces of an XML element for use in the SAX-reader or SAX-writer object. The purpose of the object is to hold values needed by the SAX parser to set the attribute values of a new XML element as it reads the element into a SAX-reader object or prepares an XML object for writing in a SAX-writer object.

SAX-attributes objects used by the SAX-reader object are automatically created, populated, and deleted in the context of the `startElement` callback handler. In that context, a programmer may use the methods and attributes of the SAX-attributes object and modify it. However, if the programmer wants to use the contents of this SAX-attributes object outside of the short scope of the `startElement` callback, then the programmer must create another SAX-attributes object outside this scope and copy the SAX-reader's SAX-attributes object to the new object.

A SAX-attributes object is an optional parameter for SAX-writer methods that create new XML elements. In this context, a programmer creates a new SAX-attributes object with the CREATE SAX-ATTRIBUTES statement. Typically, a programmer populates the object by using the COPY-SAX-ATTRIBUTES method to save the contents of an automatically created SAX-attributes object. A programmer can also use the other attributes and methods to build up the attributes list from scratch. Either way, once the SAX-attributes object is populated, it can be used as an optional parameter on the START-ELEMENT( ) method, WRITE-DATA-ELEMENT( ) method, or the WRITE-EMPTY-ELEMENT( ) method of the SAX-writer object.

You must explicitly delete all SAX-attributes objects created with the CREATE SAX-ATTRIBUTES statement.

Syntax

```
SAX-attributes-handle [ :attribute | :method ]
```

SAX-attributes-handle

A variable of type HANDLE.

attribute

An attribute of the SAX-attributes object.

method

A method of the SAX-attributes object.

Attributes

ADM-DATA attribute	HANDLE attribute	INSTANTIATING-PROCEDURE attribute
NUM-ITEMS attribute	PRIVATE-DATA attribute	TYPE attribute
UNIQUE-ID attribute	–	–

Methods

COPY-SAX-ATTRIBUTES( ) method	GET-INDEX-BY-NAMESPACE-NAME( ) method
GET-INDEX-BY-QNAME( ) method	GET-LOCALNAME-BY-INDEX( ) method
GET-QNAME-BY-INDEX( ) method	GET-TYPE-BY-INDEX( ) method
GET-TYPE-BY-NAMESPACE-NAME( ) method	GET-TYPE-BY-QNAME( ) method
GET-URI-BY-INDEX( ) method	GET-VALUE-BY-INDEX( ) method
GET-VALUE-BY-NAMESPACE-NAME( ) method	GET-VALUE-BY-QNAME( ) method
INSERT-ATTRIBUTE( ) method	REMOVE-ATTRIBUTE( ) method
UPDATE-ATTRIBUTE( ) method	–

See also

CREATE SAX-READER statement, CREATE SAX-WRITER statement, SAX-reader object handle, SAX-writer object handle

SAX-reader object handle

Provides access to the SAX parser to stream an XML document into the object. The SAX-reader object, used with the SAX interface, corresponds to the X-document object of the Document Object Model (DOM) interface, but presents a very different programming interface.

Syntax

```
SAX-reader-handle [ :attribute | :method ]
```

SAX-reader-handle

A variable of type HANDLE.

attribute

An attribute of the SAX-reader object.

method

A method of the SAX-reader object.

Attributes

ADM-DATA attribute	ENTITY-EXPANSION-LIMIT attribute	EVENT-HANDLER-OBJECT attribute

HANDLE attribute	HANDLER attribute	INSTANTIATING-PROCEDURE attribute
LOCATOR-COLUMN-NUMBER attribute	LOCATOR-LINE-NUMBER attribute	LOCATOR-PUBLIC-ID attribute
LOCATOR-SYSTEM-ID attribute	NONAMESPACE-SCHEMA-LOCATION attribute	PARSE-STATUS attribute
PRIVATE-DATA attribute	SCHEMA-LOCATION attribute	SCHEMA-PATH attribute
STRICT-ENTITY-RESOLUTION attribute	SUPPRESS-NAMESPACE-PROCESSING attribute	TYPE attribute
UNIQUE-ID attribute	VALIDATION-ENABLED attribute	–

Methods

ADD-SCHEMA-LOCATION( ) method	SAX-PARSE( ) method
SAX-PARSE-FIRST( ) method	SAX-PARSE-NEXT( ) method
SET-INPUT-SOURCE( ) method	STOP-PARSING( ) method

See also

CREATE SAX-READER statement, SAX-attributes object handle

SAX-writer object handle

A handle to the SAX-writer object. You create the handle and assign it to a handle variable with the CREATE SAX-WRITER statement.

Syntax

```
SAX-writer-handle [ :attribute | :method ]
```

SAX-writer-handle

A variable of type HANDLE.

attribute

An attribute of the SAX-writer object.

method

A method of the SAX-writer object.

Attributes

ENCODING attribute	FORMATTED attribute	FRAGMENT attribute
HANDLE attribute	INSTANTIATING-PROCEDURE attribute	STANDALONE attribute
STRICT attribute	TYPE attribute	VERSION attribute
WRITE-STATUS attribute	–	–

Methods

DECLARE-NAMESPACE( ) method	END-DOCUMENT( ) method
END-ELEMENT( ) method	INSERT-ATTRIBUTE( ) method
RESET( ) method	SET-OUTPUT-DESTINATION( ) method
START-DOCUMENT( ) method	START-ELEMENT( ) method
WRITE-CDATA( ) method	WRITE-CHARACTERS( ) method
WRITE-COMMENT( ) method	WRITE-DATA-ELEMENT( ) method
WRITE-EMPTY-ELEMENT( ) method	WRITE-ENTITY-REF( ) method
WRITE-EXTERNAL-DTD( ) method	WRITE-FRAGMENT( ) method
WRITE-PROCESSING-INSTRUCTION( ) method	–

See also

CREATE SAX-WRITER statement

SECURITY-POLICY system handle

A handle to security policy settings and operations for the current ABL session. These settings and operations support session data cryptography, creating a session registry for trusted security domains, and user authentication and validation.

Syntax

```
SECURITY-POLICY [ :attribute | :method]
```

attribute

An attribute of the SECURITY-POLICY handle.

method

A method of the SECURITY-POLICY handle.

Attributes

ALLOW-PREV-DESERIALIZATION attribute	CAN-DO-DOMAIN-SUPPORT attribute
ENCRYPTION-SALT attribute	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	PBE-HASH-ALGORITHM attribute
PBE-KEY-ROUNDS attribute	SYMMETRIC-ENCRYPTION-ALGORITHM attribute
SYMMETRIC-ENCRYPTION-IV attribute	SYMMETRIC-ENCRYPTION-KEY attribute
SYMMETRIC-SUPPORT attribute	TYPE attribute
XCODE-SESSION-KEY attribute	—

Methods

ENCODE-DOMAIN-ACCESS-CODE( ) method	GET-CLIENT( ) method
GET-SAFE-USER( ) method	LOAD-DOMAINS( ) method
LOCK-REGISTRATION( ) method	REGISTER-DOMAIN( ) method
SET-CLIENT( ) method	SET-SAFE-USER( ) method
VALIDATE-DOMAIN-ACCESS-CODE( ) method	—

Notes

- The LOAD-DOMAINS( ) method loads and locks a trusted domain registry for the session from the local domain registry of a single connected OpenEdge RDBMS. This type of registry supports domains for multi-tenant databases and is the most secure way to create a trusted domain registry for the session.
- The REGISTER-DOMAIN( ) and LOCK-REGISTRATION( ) methods allow you to build and lock an application-defined trusted domain registry for the session. This type of registry requires the non-secure exposure of critical domain information as the domain registry is being built, and is therefore less secure than using the LOAD-DOMAINS( ) method. It also does not support domains for multi-tenant databases.
- The SET-CLIENT( ) method can authenticate (or validate) and set the user identity for a security token represented by a client-principal object using a session trusted domain registry that you have previously loaded. This single method can set the user identity for both the session and existing database connections, depending on the user's domain definition and the design of your application authentication model. (Setting a database connection identity, alone, does not set the session identity.) The method can also set an audit-able session identity.The GET-CLIENT( ) method retrieves the client-principal object previously used to set the current session identity using the SET-CLIENT( ) method. For more information on managing identity, domains, and domain registries, see *Learn about Identity Management*.
- The following attributes set initial encryption values for use by the GENERATE-PBE-KEY function:
  - ENCRYPTION-SALT attribute

  - PBE-HASH-ALGORITHM attribute
  - PBE-KEY-ROUNDS attribute

- The following attributes allow you to set defaults for the ENCRYPT function and the DECRYPT function:
  - SYMMETRIC-ENCRYPTION-ALGORITHM attribute
  - SYMMETRIC-ENCRYPTION-IV attribute
  - SYMMETRIC-ENCRYPTION-KEY attribute
  - SYMMETRIC-SUPPORT attribute
- The XCODE-SESSION-KEY attribute contains the encryption key used by the COMPILE statement for the current session. This must be the same value used by the XCODE utility.
- The ENCODE-DOMAIN-ACCESS-CODE( ) method allows you to create an encrypted value suitable for storing as the domain access code in the OpenEdge database's domain records. Use the VALIDATE-DOMAIN-ACCESS-CODE( ) method to verify that the encrypted value matches the clear text value.
- The SET-SAFE-USER( ) method sets a multi-tenant database connection's current user identity to the stored safe user identity for the database. The GET-SAFE-USER( ) method returns the handle to a copy of the sealed client-principal object that represents the safe user identity for the specified database connection. Both methods are only applicable to a Progress Application Server (PAS) for OpenEdge application server.

See also

Client-principal object handle, DECRYPT function, ENCRYPT function, GENERATE-PBE-KEY function, GENERATE-PBE-SALT function, GENERATE-RANDOM-KEY function, MESSAGE-DIGEST function

SELF system handle

A handle to the built-in object or widget whose event is being handled by the current trigger or event handler.

Syntax

```
SELF [ :attribute ]
```

attribute

An attribute of the built-in object or widget whose event is being handled by the current trigger or event handler.

Attributes

The specific attributes available depend on the type of the object or widget. You can determine the object or widget type by examining the SELF:TYPE attribute.

Example

The following example uses the SELF handle to display the starting and ending positions of an object you move:

r-self.p

```
DEFINE BUTTON b_quit LABEL "Quit"
  TRIGGERS:
    ON CHOOSE QUIT.
  END.

DEFINE VARIABLE x AS CHARACTER NO-UNDO INITIAL "MOVE ME".

DEFINE FRAME move
  x NO-LABEL
  WITH SIZE 80 BY 10 TITLE "Move/Resize Widget".

ASSIGN
  x:MOVABLE    = TRUE
  x:SELECTABLE = TRUE.

DEFINE FRAME butt-frame
  b_quit
  WITH CENTERED ROW SCREEN-LINES - 1.

ON END-MOVE OF x IN FRAME move
  DISPLAY
    SELF:FRAME-ROW
    SELF:FRAME-COL
      WITH FRAME end-info CENTERED ROW 14 TITLE "End Position".

ON START-MOVE OF x IN FRAME move
  DISPLAY
    SELF:FRAME-ROW
    SELF:FRAME-COL
    WITH FRAME info CENTERED ROW 12 TITLE "Start Position".

ENABLE b_quit WITH FRAME butt-frame.
DISPLAY x WITH FRAME move.
ENABLE x WITH FRAME move.

WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame FOCUS x.
```

Notes

- In user-interface triggers, SELF is not automatically the widget that has input focus. To give input focus to the widget referenced by SELF, you must apply the ENTRY event to SELF within the trigger block. Note that you must do this for fill-in widgets whose AUTO-ZAP attribute you want to set, as in this fragment:

  ```
  DEFINE VARIABLE fname AS CHARACTER NO-UNDO FORMAT "x(30)" LABEL "Name".

  DEFINE FRAME FillFrame fname WITH SIDE-LABELS.

  ON ENTRY OF fname IN FRAME FillFrame DO:
    APPLY "ENTRY" TO SELF.
    SELF:AUTO-ZAP = TRUE.
  END.
  ```

  This makes SELF = FOCUS, which allows the new AUTO-ZAP value to take effect. For more information on the AUTO-ZAP attribute, see the Handle Attributes and Methods Reference.

- In the handler for a PROCEDURE-COMPLETE event, SELF returns the associated asynchronous request handle.
- In the event procedure of an ActiveX control, SELF returns the control-frame handle and the COM-SELF system handle returns the control-frame COM-HANDLE value.

- If referenced within a READ-RESPONSE event procedure or method then SELF is the socket handle associated with the connection that received the message.

See also

Asynchronous request object handle, COM-SELF system handle, FOCUS system handle, LAST-EVENT system handle

Server object handle

Allows you to connect and execute remote procedures on an application server or Web service.

Note: This handle does not provide direct access to an application server session context as does a SESSION handle for the current context. Rather, it provides access to a server object in the current context that allows you to connect, disconnect, and retrieve a variety of information on a connected application server.

Syntax

```
server-handle [ :attribute | :method ]
```

server-handle

A handle variable that references a server object created by the CREATE SERVER statement that, in turn, allows you to connect to and access an application server instance or a Web service application.

attribute

An attribute of the server handle.

method

A method of the server handle.

Attributes

ASYNC-REQUEST-COUNT attribute	CLIENT-CONNECTION-ID attribute[80]	FIRST-ASYNC-REQUEST attribute
FIRST-PROCEDURE attribute[80]	HANDLE attribute	INSTANTIATING-PROCEDURE attribute
LAST-ASYNC-REQUEST attribute	LAST-PROCEDURE attribute[80]	NAME attribute
NEXT-SIBLING attribute	PREV-SIBLING attribute	PRIVATE-DATA attribute

[80] For this attribute to be valid, the handle must have an application server or Web service connection (the CONNECTED( ) method must return TRUE).

REQUEST-INFO attribute	RESPONSE-INFO attribute	SOAP-VERSION attribute
SSL-SERVER-NAME attribute	SUBTYPE attribute	TYPE attribute

Methods

CANCEL-REQUESTS( ) method	CANCEL-REQUESTS-AFTER( ) method
CONNECT( ) method (AppServer)	CONNECT( ) method (Progress Application Server for OpenEdge)
CONNECT( ) method (Web service)	CONNECTED( ) method
DISCONNECT( ) method (Handle)	–

Note

For SpeedScript, as in any ABL client, a WebSpeed Agent can use a valid server handle to access and run remote procedures on an application server. However, it does not access or affect the state of any WebSpeed Transaction Server.

Server socket object handle

A handle to a server socket object. This object allows you to listen for and accept TCP/IP socket connections on a given port.

Note: Does not apply to SpeedScript programming.

Syntax

```
server-socket-handle [ :attribute | :method ]
```

server-socket-handle

A handle variable that references a server socket object created by the CREATE SERVER-SOCKET statement that, in turn, allows you to listen for and accept multiple connections on a given port.

attribute

An attribute of the server socket handle.

method

A method of the server socket handle.

Attributes

HANDLE attribute	INSTANTIATING-PROCEDURE attribute	NAME attribute
NEXT-SIBLING attribute	PREV-SIBLING attribute	PRIVATE-DATA attribute
SENSITIVE attribute	TYPE attribute	–

Methods

DISABLE-CONNECTIONS( ) method	ENABLE-CONNECTIONS( ) method
SET-CALLBACK( ) method	SET-CONNECT-PROCEDURE( ) method

Events

CONNECT event	–

Note

The server socket object is used to enable the AVM to listen to and accept new connections from socket clients; it is via the socket object that clients and servers communicate. For more information on using sockets, see *OpenEdge Programming Interfaces*.

SESSION system handle

A handle to the current ABL session object. This object allows you to read and modify the current ABL session context.

Syntax

```
SESSION [ :attribute | :method ]
```

attribute

Specifies an attribute of the SESSION system handle.

method

Specifies a method of the SESSION system handle.

Attributes

APPL-ALERT-BOXES attribute	BASE-ADE attribute
BATCH-MODE attribute	CHARSET attribute

CLIENT-TYPE attribute	CONTEXT-HELP-FILE attribute
CPCASE attribute	CPCOLL attribute
CPINTERNAL attribute	CPLOG attribute
CPPRINT attribute	CPRCODEIN attribute
CPRCODEOUT attribute	CPSTREAM attribute
CPTERM attribute	CURRENT-REQUEST-INFO attribute
CURRENT-RESPONSE-INFO attribute	DATA-ENTRY-RETURN attribute
DATE-FORMAT attribute	DEBUG-ALERT attribute
DISPLAY-TIMEZONE attribute	DISPLAY-TYPE attribute
ERROR-STACK-TRACE attribute	EXECUTION-LOG attribute
EXIT-CODE attribute	FIRST-BUFFER attribute
FIRST-CHILD attribute	FIRST-DATASET attribute
FIRST-DATA-SOURCE attribute	FIRST-FORM attribute
FIRST-OBJECT attribute	FIRST-PROCEDURE attribute
FIRST-QUERY attribute	FIRST-SERVER attribute
FIRST-SERVER-SOCKET attribute	FIRST-SOCKET attribute
FRAME-SPACING attribute	HANDLE attribute
HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute
ICFPARAMETER attribute	IMMEDIATE-DISPLAY attribute
INHERIT-BGCOLOR attribute	INHERIT-FGCOLOR attribute
INSTANTIATING-PROCEDURE attribute	LAST-CHILD attribute
LAST-FORM attribute	LAST-OBJECT attribute
LAST-PROCEDURE attribute	LAST-SERVER attribute
LAST-SERVER-SOCKET attribute	LAST-SOCKET attribute
LOCAL-VERSION-INFO attribute	MULTITASKING-INTERVAL attribute
NEXT-SIBLING attribute	NUMERIC-DECIMAL-POINT attribute
NUMERIC-FORMAT attribute	NUMERIC-SEPARATOR attribute

PARAMETER attribute	PIXELS-PER-COLUMN attribute
PIXELS-PER-ROW attribute	PRINTER-CONTROL-HANDLE attribute
PRINTER-HDC attribute	PRINTER-NAME attribute
PRINTER-PORT attribute	PROXY-PASSWORD attribute
PROXY-USERID attribute	REMOTE attribute
SCHEMA-CHANGE attribute	SERVER-CONNECTION-BOUND attribute
SERVER-CONNECTION-BOUND-REQUEST attribute	SERVER-CONNECTION-CONTEXT attribute
SERVER-CONNECTION-ID attribute	SERVER-OPERATING-MODE attribute
STARTUP-PARAMETERS attribute	STREAM attribute
SUPER-PROCEDURES attribute	SUPPRESS-WARNINGS attribute
SUPPRESS-WARNINGS-LIST attribute	SYSTEM-ALERT-BOXES attribute
THREE-D attribute	TIME-SOURCE attribute
TIMEZONE attribute	TOOLTIPS attribute
TYPE attribute	V6DISPLAY attribute
WC-ADMIN-APP attribute	WIDTH-CHARS attribute
WIDTH-PIXELS attribute	WINDOW-SYSTEM attribute
WORK-AREA-HEIGHT-PIXELS attribute	WORK-AREA-WIDTH-PIXELS attribute
WORK-AREA-X attribute	WORK-AREA-Y attribute
YEAR-OFFSET attribute	TEMP-DIRECTORY attribute

Methods

ADD-SUPER-PROCEDURE( ) method	EXPORT( ) method
GET-PRINTERS( ) method	GET-WAIT-STATE( ) method
REMOVE-SUPER-PROCEDURE( ) method	SET-NUMERIC-FORMAT( ) method
SET-WAIT-STATE( ) method	–

Example

The following example uses the SESSION:IMMEDIATE-DISPLAY attribute. When dumping or loading records from the database, the procedure displays a running count of records. If IMMEDIATE-DISPLAY is false, no value is shown until all records are dumped or loaded. At that point, the total is shown. To prevent this, IMMEDIATE-DISPLAY is set to true just before the dump or load and then reset to false afterwards.

r-dstrig.p

```
DEFINE VARIABLE i AS INTEGER NO-UNDO.

DEFINE SUB-MENU file
  MENU-ITEM viewit LABEL "&View Data"
  MENU-ITEM dumpit LABEL "&Dump Data"
  MENU-ITEM loadit LABEL "&Load Data".
  MENU-ITEM exit   LABEL "E&xit".
DEFINE MENU mbar MENUBAR
  SUB-MENU file LABEL "&File".

DEFINE BUTTON b_more LABEL "Next".
DEFINE BUTTON b_exit LABEL "Cancel".

DEFINE FRAME cust-frame
  Customer.CustNum SKIP
  Customer.Name SKIP
  Customer.Phone SKIP
  b_more b_exit
  WITH CENTERED SIDE-LABELS ROW 3.

DEFINE STREAM cust.

PAUSE 0 BEFORE-HIDE.

ON CHOOSE OF b_exit IN FRAME cust-frame DO:
  HIDE FRAME cust-frame NO-PAUSE.
  DISABLE ALL WITH FRAME cust-frame.
  LEAVE.
END.

ON CHOOSE OF b_more IN FRAME cust-frame DO:
  FIND NEXT Customer NO-LOCK NO-ERROR.
  IF NOT AVAILABLE Customer THEN
    RETURN.
  DISPLAY Customer.CustNum Customer.Name Customer.Phone
    WITH FRAME cust-frame.
END.

ON CHOOSE OF MENU-ITEM viewit DO:
  ENABLE ALL WITH FRAME cust-frame.
  FIND FIRST Customer NO-LOCK NO-ERROR.
  DISPLAY Customer.CustNum Customer.Name Customer.Phone
    WITH FRAME cust-frame.
  APPLY "ENTRY" TO b_more.
END.

ON CHOOSE OF MENU-ITEM dumpit DO:
  DISABLE TRIGGERS FOR DUMP OF Customer.
  ix = 1.
  SESSION:IMMEDIATE-DISPLAY = TRUE.

  OUTPUT STREAM cust TO "customer.d".
  FOR EACH Customer NO-LOCK:
    EXPORT STREAM cust Customer.
    DISPLAY ix LABEL "Records Processed"
      WITH FRAME rec-info SIDE-LABELS ROW SCREEN-LINES / 2 CENTERED.
    ix = ix + 1.
  END.
  SESSION:IMMEDIATE-DISPLAY = FALSE.
```

```
  OUTPUT STREAM cust CLOSE.
END.

ON CHOOSE OF MENU-ITEM loadit DO:
  DISABLE TRIGGERS FOR LOAD OF Customer.
  INPUT FROM "customer.d".
  SESSION:IMMEDIATE-DISPLAY = TRUE.
  REPEAT:
    CREATE Customer.
    IMPORT Customer.
    DISPLAY ix LABEL "Records Processed"
      WITH FRAME rec-info SIDE-LABELS ROW SCREEN-LINES / 2 CENTERED.
    ix = ix + 1.
  END.
  INPUT CLOSE.
  SESSION:IMMEDIATE-DISPLAY = FALSE.
END.

IF NOT RETRY THEN
ASSIGN CURRENT-WINDOW:MENUBAR = MENU mbar:HANDLE
       CURRENT-WINDOW:VISIBLE = TRUE.

WAIT-FOR CHOOSE OF MENU-ITEM exit.
```

Notes

- Several attributes of the SESSION handle control the execution of ABL code during the current ABL session. This means that the SESSION handle controls the behavior of any code that you are developing and testing, and the OpenEdge ADE toolset. While the tools of the OpenEdge ADE monitor and set the attributes of the SESSION handle to meet their needs, it is possible that the execution of a procedure that sets attributes of the SESSION handle may affect the display and behavior of the OpenEdge ADE toolset.
- The FIRST-PROCEDURE and LAST-PROCEDURE attributes are set or reset when you create or delete the first or last persistent procedure in a session. You can use procedure attributes to navigate the procedure entries, reference information, and manage the user interface for each persistent procedure in the procedure chain accessed by FIRST-PROCEDURE and LAST-PROCEDURE.

  For more information on the attributes of procedure handles, see the Procedure object handle reference entry. For information on creating a persistent procedure, see the RUN statement. For information on deleting a persistent procedure, see the DELETE PROCEDURE statement reference entry.

- The FIRST-SERVER and LAST-SERVER attributes are set or reset when you create or delete the first or last server handle in a session. You can use server handle attributes and methods to navigate the current chain of server handles, connect to a running application server, reference information on a connected application server, access remote persistent procedures running on a connected application server, and disconnect from a connected application server for each server handle in the chain accessed by FIRST-SERVER and LAST-SERVER.

  For more information on the attributes and methods of server handles, see the Server object handle reference entry. For information on creating server handles, see the CREATE SERVER statement reference entry.

- Setting the IMMEDIATE-DISPLAY attribute to TRUE can significantly slow performance. However, some code segments may not execute properly with IMMEDIATE-DISPLAY set to FALSE. If a segment of code requires that IMMEDIATE-DISPLAY is TRUE, you should set the attribute to TRUE immediately before the code segment and change it back to FALSE immediately after the segment.
- In Windows, when execution is blocked for input (by a WAIT-FOR statement, for example), the AVM listens for messages from the windowing system. This allows the AVM to multitask properly with other Windows applications. However, if your ABL application performs long processing without blocking for input, then it may not multitask properly because the AVM does not automatically check for messages from the windowing system. To force the AVM to poll for windowing system messages during this time, you can set the

MULTITASKING-INTERVAL attribute to a non-zero value. The lower the value, the more often the AVM checks for messages. This may decrease ABL performance. The maximum value is 9999. A value of 0 inhibits polling until ABL blocks for input.

If you set MULTITASKING-INTERVAL to a non-zero value for a code segment, reset it to 0 immediately after that code.

- The AVM sets the TEMP-DIRECTORY attribute to the value you specify for the Temporary Directory (-T) parameter. If you omit the -T parameter, TEMP-DIRECTORY is set to your current working directory.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.
- Use the SET-WAIT-STATE method to prevent user and system input, and provide visual feedback during a long computation or other background process. The value you pass determines the type of wait message or cursor the windowing system displays for the user. Passing the value "" to SET-WAIT-STATE ends the wait state. Use this method only for long computations or other processes that force the user to wait significantly longer than the usual response time.
- If you set a wait state for your application, the AVM automatically ends the wait state if it displays an alert box, a dialog box, or message update.
- For SpeedScript, the invalid attributes are: APPL-ALERT-BOXES, CONTEXT-HELP-FILE, DATA-ENTRY-RETURN, FIRST-CHILD, HEIGHT-PIXELS, LAST-CHILD, PARAMETER, PIXELS-PER-COLUMN, PIXELS-PER-ROW, SUPPRESS-WARNINGS, SUPPRESS-WARNINGS-LIST, SYSTEM-ALERT-BOXES, THREE-D, TOOLTIPS, V6DISPLAY, WIDTH-PIXELS. The GET-PRINTERS( ) method is invalid for SpeedScript.
- To access the list of all forms created in a session, use the SESSION handle FIRST-FORM and LAST-FORM attributes along with the NextForm and PrevForm properties of the Progress.Windows.IForm interface. In addition, to .NET forms, this list also contains Progress.Windows.FormProxy object references to all ABL windows created in a session, allowing you to manage .NET forms and ABL windows in a common manner. The SESSION handle FIRST-CHILD attribute, NEXT-SIBLING attribute, LAST-CHILD attribute, and PREV-SIBLING attribute only reference the list of ABL windows in a session.
- EXIT-CODE attribute can be used to set an exit code, from within the application, so it's available to the operating system when the AVM process ends. However, the value returned may be modified by the shell the user is calling the AVM process from.

See also

NextForm property, PrevForm property, Progress.Windows.FormProxy class, Progress.Windows.Form class, Progress.Windows.IForm interface

SOAP-fault object handle

A handle to a SOAP-fault object. A SOAP-fault object contains information specific to a SOAP fault.

Note: Does not apply to SpeedScript programming.

Syntax

```
soap-fault-handle [ :attribute ]
```

soap-fault-handle

A handle variable that references a SOAP-fault object.

attribute

An attribute of the SOAP-fault object.

Attributes

HANDLE attribute	INSTANTIATING-PROCEDURE attribute	SOAP-FAULT-ACTOR attribute
SOAP-FAULT-CODE attribute	SOAP-FAULT-DETAIL attribute	SOAP-FAULT-STRING attribute
SOAP-FAULT-MISUNDERSTOOD-HEADER attribute	SOAP-FAULT-NODE attribute	SOAP-FAULT-ROLE attribute
SOAP-FAULT-SUBCODE attribute	TYPE attribute	–

Notes

- When the AVM detects a SOAP fault message, it converts the SOAP fault message to an ABL error message and creates a SOAP-fault object (identified by the ERROR-OBJECT-DETAIL attribute on the ERROR-STATUS system handle). A SOAP-fault object exists only as long as its related ERROR-STATUS entry (that is, until the execution of another statement containing the NO-ERROR option).
- Use the SOAP-FAULT-DETAIL object handle to access the SOAP fault message detail.

See also

ERROR-STATUS system handle, SOAP-fault-detail object handle

SOAP-fault-detail object handle

A handle to a SOAP-fault-detail object.

Note: Does not apply to SpeedScript programming.

Syntax

```
soap-fault-detail-handle [ :attribute | :method ]
```

soap-fault-detail-handle

A handle variable that references a SOAP-fault-detail object.

attribute

An attribute of the SOAP-fault-detail object.

method

A method of the SOAP–FAULT-DETAIL object.

Attributes

HANDLE attribute	INSTANTIATING-PROCEDURE attribute	TYPE attribute

Methods

GET-NODE( ) method	GET-SERIALIZED( ) method

Notes

You can use the GET-NODE( ) method to get an X-noderef object handle that refers to the XML that underlies a SOAP-fault-detail object. The application can then use this X-noderef object handle to access the underlying XML. The only restriction is that the application cannot use the X-noderef object handle retrieved from the SOAP-fault-detail object to access the X-document associated with the SOAP-fault object.

SOAP-header object handle

A handle to a SOAP-header object. The object maintains a list of entries. Each entry is associated with an existing SOAP-header-entryref object. A SOAP-header object is passed as an input parameter to a response callback procedure and as an output parameter to a request callback procedure.

Note: Does not apply to SpeedScript programming.

Syntax

```
soap-header-handle [ :attribute | :method ]
```

soap-header-handle

A handle variable that references a SOAP-header object.

attribute

An attribute of the SOAP-header object.

method

A method of the SOAP-header object.

Attributes

ADM-DATA attribute	HANDLE attribute	INSTANTIATING-PROCEDURE attribute
NAME attribute	NUM-HEADER-ENTRIES attribute	PRIVATE-DATA attribute
TYPE attribute	UNIQUE-ID attribute	–

Methods

ADD-HEADER-ENTRY( ) method	GET-HEADER-ENTRY( ) method

Note

The SOAP-header object is either implicitly created by ABL or explicitly created by the application using the CREATE SOAP-HEADER statement. In either case, the application is responsible for deleting this object. Use the DELETE OBJECT statement to delete a SOAP-header object and its underlying XML.

See also

CREATE SOAP-HEADER statement, SOAP-header-entryref object handle

SOAP-header-entryref object handle

A handle to a SOAP-header-entryref object.

Note: Does not apply to SpeedScript programming.

Syntax

```
soap-header-entryref-handle [ :attribute | :method ]
```

soap-header-entryref-handle

A handle variable that references a SOAP-header-entryref object.

attribute

An attribute of the SOAP-header-entryref object.

method

A method of the SOAP-header-entryref object.

Attributes

ACTOR attribute	ADM-DATA attribute	HANDLE attribute

INSTANTIATING-PROCEDURE attribute	LOCAL-NAME attribute	MUST-UNDERSTAND attribute
NAME attribute	NAMESPACE-URI attribute	PRIVATE-DATA attribute
ROLE attribute	TYPE attribute	UNIQUE-ID attribute

Methods

DELETE-HEADER-ENTRY( ) method	GET-NODE( ) method
GET-SERIALIZED( ) method	SET-ACTOR( ) method
SET-MUST-UNDERSTAND( ) method	SET-NODE( ) method
SET-ROLE( ) method	SET-SERIALIZED( ) method

Notes

- The application is responsible for deleting this object. Use the DELETE OBJECT statement to delete a SOAP-header-entryref object without deleting its underlying XML. To delete the XML underlying the SOAP-header-entryref object, without deleting the object, use the DELETE-HEADER-ENTRY() method.
- You can use the GET-NODE( ) method to get an X-noderef object handle that refers to the XML that underlies a SOAP-header-entryref object. The application can then use this X-noderef object handle to access the underlying XML. The only restriction is that the application cannot use the X-noderef object handle retrieved from the SOAP-header-entryref object to access the X-document associated with the SOAP-header object.

See also

CREATE SOAP-HEADER-ENTRYREF statement, SOAP-header object handle

Socket object handle

A handle to a socket object. This object allows you to read or write data on a TCP/IP socket and to perform other TCP/IP socket actions.

Syntax

```
socket-handle [ :attribute | :method ]
```

socket-handle

A handle variable that references a socket object created by the CREATE SOCKET statement and that allows you to connect to, read from and write to a socket.

attribute

An attribute of the socket handle.

method

A method of the socket handle.

Attributes

BYTES-READ attribute	BYTES-WRITTEN attribute	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	LOCAL-HOST attribute	LOCAL-PORT attribute
NAME attribute	NEXT-SIBLING attribute	PREV-SIBLING attribute
PRIVATE-DATA attribute	REMOTE-HOST attribute	REMOTE-PORT attribute
SENSITIVE attribute	SSL-SERVER-NAME attribute	TYPE attribute

Methods

CONNECT( ) method (Socket object)	CONNECTED( ) method
DISCONNECT( ) method (Handle)	GET-BYTES-AVAILABLE( ) method
GET-SOCKET-OPTION( ) method	READ( ) method (Socket)
SET-CALLBACK( ) method	SET-READ-RESPONSE-PROCEDURE( ) method
SET-SOCKET-OPTION( ) method	WRITE( ) method (Socket)

Events

READ-RESPONSE event	–

Note

The server socket object is used to enable connections from socket clients; it is via the socket object that clients and servers communicate. For more information on using sockets, see *OpenEdge Programming Interfaces*.

SOURCE-PROCEDURE system handle

A handle to the procedure file that contains the original invocation (RUN statement or function invocation) of the current internal procedure or user-defined function.

Syntax

```
SOURCE-PROCEDURE [ :attribute | :method ]
```

attribute

An attribute of the SOURCE-PROCEDURE handle.

method

A method of the SOURCE-PROCEDURE handle.

Attributes

The SOURCE-PROCEDURE handle supports all the attributes of the procedure object handle. For a list of these attributes, see the reference entry for the Procedure object handle in this section.

Methods

The SOURCE-PROCEDURE handle supports all the methods of the procedure object handle. For a list of these methods, see the reference entry for the Procedure object handle in this section.

Example

The following scenarios illustrate using SOURCE-PROCEDURE without procedure overriding, with procedure overriding, and with super and non-super RUNs:

Scenario 1: Using SOURCE-PROCEDURE without procedure overriding

The following scenario uses SOURCE-PROCEDURE without procedure overriding:

1. A and B are handles of procedure files running persistently.
2. proc1 is an internal procedure that resides in B.
3. A says "RUN proc1 IN B," which runs B's proc1.

In this scenario:

- The original run statement for proc1 occurs in Step 3.
- Within B's proc1 (and within any proc1 that runs as a result of its original RUN statement), SOURCE-PROCEDURE is A.

Scenario 2: Using SOURCE-PROCEDURE with procedure overriding

The following scenario uses SOURCE-PROCEDURE with procedure overriding:

1. A, B, and C, and X are handles of procedure files running persistently.
2. B is a super procedure of A, and C is a super procedure of B.
3. proc1 is an internal procedure different versions of which reside in A, B, and C.

Note: This is an example of procedure overriding.

---

4. X says "RUN proc1 IN A," which runs A's proc1.
5. A's proc1 says "RUN SUPER," which runs B's proc1.
6. B's proc1 says "RUN SUPER," which runs C's proc1.

In this scenario:

- The original run statement for proc1 occurs in Step 4.
- Within any version of proc1 that runs as a result of its original RUN statement, SOURCE-PROCEDURE is X.

Scenario 3: With SUPER and non-SUPER RUNs

The following scenario shows how the value of SOURCE-PROCEDURE changes when a non-super RUN occurs:

1. A, B, and C are handles of procedure files running persistently.
2. B is a super procedure of A, and C is a super procedure of B.
3. proc1 is an internal procedure different versions of which reside in A, B, and C.
4. proc2 is an internal procedure different versions of which reside in A, B, and C.
5. A says "RUN proc1," which runs A's proc1.
6. A's proc1 says "RUN SUPER," which runs B's proc1.
7. B's proc1 says "RUN SUPER," which runs C's proc1.

   Note: At this point, within any proc1 that runs as a result of its original RUN statement, the value of SOURCE-PROCEDURE is A.

8. C's proc1 says "RUN proc2," which runs C's proc2.

   Note: This is a non-super RUN.

In this scenario:

- The original RUN statement for proc1 occurs in Step 5.
- Within any proc1 that runs as a result of its original RUN statement, SOURCE-PROCEDURE is A.
- The original RUN statement for proc2 occurs in Step 8.
- Within any proc2 that runs as a result of its original RUN statement, SOURCE-PROCEDURE is C.

For a sample program that uses SOURCE-PROCEDURE, see the reference entry for the RUN SUPER statement.

Notes

- You can use SOURCE-PROCEDURE in applications that do not use super procedures.
- In the main block of a procedure, the value of SOURCE-PROCEDURE is the handle of the procedure that ran the current ABL source code or r-code file. This allows any ABL program to identify its caller, and to perform a "callback" to its caller.
- If an ABL or other client runs a procedure on an application server, then in the procedure running on the application server, the value of SOURCE-PROCEDURE is the Unknown value (?).

See also

ADD-SUPER-PROCEDURE( ) method, REMOVE-SUPER-PROCEDURE( ) method, RUN SUPER statement, SUPER function, SUPER-PROCEDURES attribute, TARGET-PROCEDURE system handle

Stream object handle

The AVM creates a stream object when an application first tries to get the handle to a given stream. The AVM automatically deletes the stream object when the procedure or class that defines the stream terminates.

Syntax

```
STREAM stream-name:attribute | stream-handle:attribute
```

stream-name

The valid name of a stream. If the stream name is invalid, ABL raises a compiler error.

attribute

An attribute of the Stream object.

Attributes

ADM-DATA attribute	NAME attribute
HANDLE attribute	PRIVATE-DATA attribute
INSTANTIATING-PROCEDURE attribute	TYPE attribute

Example

This example shows how the HANDLE attribute of the stream object is assigned to `hStream`:

```
DEFINE VARIABLE hStream AS HANDLE NO-UNDO.

DEFINE STREAM myStream.

hStream = STREAM mySTream:HANDLE.
```

See also

DEFINE STREAM statement

TARGET-PROCEDURE system handle

From within an internal procedure: A handle to the procedure file mentioned, explicitly or implicitly, by the original RUN statement that invoked (perhaps through a chain of super procedures) the current internal procedure.

From within a user-defined function: A handle to the procedure file mentioned, explicitly or implicitly, by the original function invocation that invoked (perhaps through a chain of super versions of functions) the current user-defined function.

Syntax

```
TARGET-PROCEDURE [ :attribute | :method ]
```

attribute

An attribute of the TARGET-PROCEDURE handle.

method

A method of the TARGET-PROCEDURE handle.

Attributes

The TARGET-PROCEDURE handle supports all the attributes of the procedure object handle. For a list of these attributes, see the reference entry for the Procedure object handle in this section.

Methods

The TARGET-PROCEDURE handle supports all the methods of the procedure object handle. For a list of these methods, see the reference entry for the Procedure object handle in this section.

Examples

The following scenarios illustrate using TARGET-PROCEDURE without procedure overriding, with procedure overriding, and with super and non-super RUNs:

Scenario 1: Using TARGET-PROCEDURE without procedure overriding

The following scenario uses TARGET-PROCEDURE without procedure overriding:

1. A and B are handles of procedure files running persistently.
2. proc1 is an internal procedure that resides in B.
3. A says "RUN proc1 IN B," which runs B's proc1.

In this scenario:

- The original RUN statement for proc1 occurs in Step 3.
- Within proc1 (and any proc1 that runs as a result its original RUN statement), the value of TARGET-PROCEDURE is B.

Scenario 2: Using TARGET-PROCEDURE with procedure overriding

The following scenario uses TARGET-PROCEDURE with procedure overriding:

1. A, B, and C, and X are handles of procedure files running persistently.
2. B is a super procedure of A, and C is a super procedure of B.
3. proc1 is an internal procedure, different versions of which reside in A, B, and C.

Note: This is an example of procedure overriding.

---

4. X says "RUN proc1 in A," which runs A's proc1.
5. A's proc1 says "RUN SUPER," which runs B's proc1.
6. B's proc1 says "RUN SUPER," which runs C's proc1.

In this scenario:

- The original RUN statement for proc1 occurs in Step 4.
- Within any version of proc1 that runs as a result of the original RUN statement, the value of TARGET-PROCEDURE is A.

Scenario 3: With procedure overriding and additional complications

The following scenario uses TARGET-PROCEDURE with procedure overriding:

1. A, B, and C, and X are handles of procedure files running persistently.
2. B is a super procedure of A, and C is a super procedure of B.
3. proc1 is an internal procedure, different versions of which reside in B and C.

   Note: proc1 does not reside in A.

4. X says "RUN proc1 in A," which runs B's proc1 (since A has no proc1 and B is a super procedure of A).
5. B's proc1 says "RUN SUPER," which runs C's proc1.

In this scenario:

- The original RUN statement for proc1 occurs in Step 4.
- Within any version of proc1 that runs as a result of its original RUN statement, the value of TARGET-PROCEDURE is A.

Scenario 4: With SUPER and non-SUPER RUNs

The following scenario shows how the value of TARGET-PROCEDURE changes when a non-super RUN occurs:

1. A, B, and C are handles of procedure files running persistently.
2. B is a super procedure of A, and C is a super procedure of B.
3. proc1 is an internal procedure different versions of which reside in A, B, and C.
4. proc2 is an internal procedure different versions of which reside in A, B, and C.
5. A says "RUN proc1," which runs A's proc1.
6. A's proc1 says "RUN SUPER," which runs B's proc1.

   Note: At this point, within any version of proc1 that runs as a result of its original RUN statement, the value of TARGET-PROCEDURE is A.

7. B's proc1 says "RUN proc2," which runs B's proc2.

   Note: This is a non-super RUN.

In this scenario:

- The original RUN statement for proc2 occurs in Step 7.
- Within any proc2 that runs as a result of its original RUN statement, the value of TARGET-PROCEDURE is B.

For a sample program that uses TARGET-PROCEDURE, see the reference entry for the RUN SUPER statement.

Notes

- You can use TARGET-PROCEDURE in applications that do not use super procedures.
- The value of TARGET-PROCEDURE becomes THIS-PROCEDURE in the following places:
  - Within the main block of a procedure file.
  - Within an internal procedure that is not a super version of another internal procedure.
  - Within a user-defined function that is not a super version of another user-defined function.

See also

ADD-SUPER-PROCEDURE( ) method, REMOVE-SUPER-PROCEDURE( ) method, RUN SUPER statement, SOURCE-PROCEDURE system handle, SUPER function, SUPER-PROCEDURES attribute, THIS-PROCEDURE system handle

Temp-table object handle

A handle to a temp-table object. A temp-table object handle corresponds to an underlying ABL temp-table, which can be static or dynamic. A static temp-table is one you define at compile time with the DEFINE TEMP-TABLE statement. A dynamic temp-table is one you create at run time with the CREATE TEMP-TABLE statement.

Syntax

```
temp-table-handle [ :attribute | :method ]
```

temp-table-handle

An item of type HANDLE representing a handle to a temp-table object.

attribute

An attribute of the temp-table object.

method

A method of the temp-table object.

Attributes

ADM-DATA attribute	AFTER-TABLE attribute	BEFORE-TABLE attribute
DATA-SOURCE-MODIFIED attribute	DEFAULT-BUFFER-HANDLE attribute	DYNAMIC attribute
ERROR attribute	ERROR-STRING attribute	HANDLE attribute
HAS-RECORDS attribute	INSTANTIATING-PROCEDURE attribute	MIN-SCHEMA-MARSHAL attribute

NAME attribute	NAMESPACE-PREFIX attribute	NAMESPACE-URI attribute
NO-SCHEMA-MARSHAL attribute	NUM-REFERENCES attribute	ORIGIN-HANDLE attribute
PREPARED attribute	PRIMARY attribute	PRIVATE-DATA attribute
REJECTED attribute	SCHEMA-MARSHAL attribute	SERIALIZE-NAME attribute
TRACKING-CHANGES attribute	TYPE attribute	UNDO attribute
UNIQUE-ID attribute	XML-NODE-NAME attribute	–

Methods

ADD-FIELDS-FROM( ) method	ADD-INDEX-FIELD( ) method
ADD-LIKE-FIELD( ) method	ADD-LIKE-INDEX( ) method
ADD-NEW-FIELD( ) method	ADD-NEW-INDEX( ) method
CLEAR( ) method (Temp-Table Object Handle)	COPY-TEMP-TABLE( ) method
CREATE-LIKE( ) method	CREATE-LIKE-SEQUENTIAL( ) method
EMPTY-TEMP-TABLE( ) method	READ-JSON( ) method
READ-XML( ) method	READ-XMLSCHEMA( ) method
TEMP-TABLE-PREPARE( ) method	WRITE-JSON( ) method
WRITE-XML( ) method	WRITE-XMLSCHEMA( ) method

Example

The following code fragment demonstrates the creation, definition and use of a temp-table object:

```
DEFINE VARIABLE tth             AS HANDLE NO-UNDO.
DEFINE VARIABLE bh              AS HANDLE NO-UNDO.
DEFINE VARIABLE qh              AS HANDLE NO-UNDO.
DEFINE VARIABLE buf-cust-handle AS HANDLE NO-UNDO.

/* Get db table handle as usual */
buf-cust-handle = BUFFER Customer:HANDLE.
/* Create an "empty" undefined temp-table */
CREATE TEMP-TABLE tth.
/* Give it Customer's fields and indexes */
tth:CREATE-LIKE(buf-cust-handle).
/* Give it a single extra field */
tth:ADD-NEW-FIELD("f1","integer").
/* No more fields or indexes will be added to custx */
tth:TEMP-TABLE-PREPARE("custx").
/* Get the buffer handle for the temp-table */
bh = tth:DEFAULT-BUFFER-HANDLE.
/* Populate the table from Customer table */
FOR EACH Customer NO-LOCK:
  bh:BUFFER-CREATE.
  bh:BUFFER-COPY(buf-cust-handle).
END.
/* Run a query to access it*/
CREATE QUERY qh.
qh:SET-BUFFERS(bh).
qh:QUERY-PREPARE("FOR EACH custx WHERE . . .").
. . .
```

Notes

- The temp-table object has three states, CLEAR, UNPREPARED and PREPARED. The temp-table is in a CLEAR state either when the temp-table is first created or immediately after the CLEAR( ) method is applied. The temp-table is in an UNPREPARED state during the period after the first definitional method has been applied and before the TEMP-TABLE-PREPARE( ) method is applied. The temp-table is in a PREPARED state after the TEMP-TABLE-PREPARE( ) method has been applied.
- The user can discern whether the temp-table is in an UNPREPARED or PREPARED state by using the PREPARED attribute.

See also

Buffer object handle, CREATE TEMP-TABLE statement, DEFINE TEMP-TABLE statement, ProDataSet object handle

THIS-PROCEDURE system handle

A handle to the procedure object for the currently executing procedure. This object allows you to read and modify the context of the current procedure.

Syntax

```
THIS-PROCEDURE [ :attribute | :method ]
```

attribute

An attribute of THIS-PROCEDURE.

method

Specifies a method of THIS-PROCEDURE.

Attributes

The THIS-PROCEDURE handle supports all the attributes of the procedure object handle. For a list of these attributes, see the reference entry for the Procedure object handle in this section.

Methods

The THIS-PROCEDURE handle supports all the methods of the procedure object handle. For a list of these methods, see the reference entry for the Procedure object handle in this section.

Example

The following procedure is designed to run both persistently and non-persistently. It sets up a query on the Customer table of the `Sports2020` database that is selectable by `Name` or `Balance`.

r-thispr.p

```
DEFINE VARIABLE custwin AS HANDLE.

DEFINE QUERY custq FOR Customer.
DEFINE BROWSE custb QUERY custq
  DISPLAY Customer.Name Customer.Balance Customer.Phone WITH 10 DOWN.

DEFINE BUTTON bName    LABEL "Query on Name".
DEFINE BUTTON bBalance LABEL "Query on Balance".
DEFINE BUTTON bCancel  LABEL "Cancel".

DEFINE FRAME CustFrame custb SKIP
  bName bBalance bCancel.

ON CHOOSE OF bName IN FRAME CustFrame DO:
  custwin:TITLE = "Customers by Name".
  OPEN QUERY custq FOR EACH Customer BY Customer.Name.
END.

ON CHOOSE OF bBalance IN FRAME CustFrame DO:
  custwin:TITLE = "Customers by Balance".
  OPEN QUERY custq FOR EACH Customer BY Customer.Balance DESCENDING.
END.

IF THIS-PROCEDURE:PERSISTENT THEN DO:
  THIS-PROCEDURE:PRIVATE-DATA = "Customer Browse".
  CREATE WIDGET-POOL.
END.

CREATE WINDOW custwin ASSIGN
  TITLE            = "Customer Browser"
  SCROLL-BARS      = FALSE
  MAX-HEIGHT-CHARS = FRAME CustFrame:HEIGHT-CHARS
  MAX-WIDTH-CHARS  = FRAME CustFrame:WIDTH-CHARS.

THIS-PROCEDURE:CURRENT-WINDOW = custwin.

ENABLE ALL WITH FRAME CustFrame.

IF THIS-PROCEDURE:PERSISTENT THEN
```

```
  ON CHOOSE OF bCancel IN FRAME CustFrame DO:
    RUN destroy-query.
  END.
ELSE
  WAIT-FOR CHOOSE OF bCancel IN FRAME CustFrame.

PROCEDURE destroy-query:
  DELETE PROCEDURE THIS-PROCEDURE.
  DELETE WIDGET-POOL.
END PROCEDURE.
```

The procedure uses the THIS-PROCEDURE handle to distinguish between persistent and non-persistent instances of execution. When `r-thispr.p` is persistent (THIS-PROCEDURE:PERSISTENT = TRUE), it:

- Sets the PRIVATE-DATA attribute to help identify it to other procedures.
- Creates a private widget pool to maintain its dynamic window for as long as the procedure instance persists.
- Defines a trigger to delete the procedure when it is terminated. Note that the trigger calls the internal procedure destroy-query, which can be executed by other external procedures to delete `r-thispr.p` when it is persistent. This destroy-query routine references the THIS-PROCEDURE handle to delete its persistent parent. It also deletes the widget pool that maintains the dynamic window.

When `r-thispr.p` is non-persistent (THIS-PROCEDURE:PERSISTENT = FALSE), it invokes a WAIT-FOR statement rather than defining a trigger to terminate the procedure. It does not need to create a widget pool or maintain any other persistent context.

Note that because both persistent and non-persistent instances of this procedure use a dynamic window separate from the default window, `r-thispr.p` assigns the window's handle to the procedure's CURRENT-WINDOW attribute. This makes the dynamic window current whether or not the procedure is persistent. However, when the procedure is persistent, the CURRENT-WINDOW attribute keeps the dynamic window current while other procedures execute using different windows. Because the persistent procedure has its own current window, its triggers and internal procedures do not have to reset the current window every time they execute.

Notes

- By determining if the current procedure is persistent, you can decide whether or not to perform certain actions. An action that you might perform during a non-persistent procedure is to execute a WAIT-FOR statement to provide input-blocking. Actions that you might execute during a persistent procedure include creating a new window to parent all other widgets created in the procedure, or maintaining an unscoped record buffer that lasts as long as the procedure persists.
- To create an instance of a persistent procedure, use the PERSISTENT option of the RUN statement. For an example, see the reference entry for the Procedure object handle in this section.
- If THIS-PROCEDURE is persistent and the NEXT-SIBLING or PREV-SIBLING attributes are invalid, THIS-PROCEDURE specifies the last or first persistent procedure instance (respectively) in the session persistent procedure chain. To check the validity of these attributes, use the VALID-HANDLE function.

See also

Procedure object handle, RUN statement, SESSION system handle, VALID-HANDLE function

Transaction object handle

Provides access to the current transaction object. This object allows you to query status of and control the current transaction context.

Syntax

```
transaction-handle [ :attribute | :method ]
```

transaction-handle

A handle variable whose value you return from the TRANSACTION attribute on a procedure object handle.

attribute

An attribute of the transaction handle.

method

A method of the transaction handle.

Attributes

DEFAULT-COMMIT attribute	HANDLE attribute	INSTANTIATING-PROCEDURE attribute
IS-OPEN attribute	TRANS-INIT-PROCEDURE attribute	TYPE attribute

Methods

SET-COMMIT( ) method	SET-ROLLBACK( ) method

Notes

- In an application server session, if a transaction initiating procedure is active, this handle allows you to control the (automatic) transaction using all of the supported attributes and methods. For more information on automatic transactions, see the TRANSACTION-MODE AUTOMATIC statement reference entry.
- In an ABL client session or in an application server session with no active transaction initiating procedure, only the IS-OPEN attribute is available.
- If a transaction initiating procedure is deleted, any open transaction is committed or rolled back according to the value of the DEFAULT-COMMIT attribute.
- The value of this attribute remains the same (references the same transaction context) for the duration of an ABL session. This is true in the following instances:
  - Whether or not a transaction is opened or closed.
  - In an application server session, whether or not a transaction initiating procedure is created or deleted.

See also

TRANSACTION-MODE AUTOMATIC statement

WEB-CONTEXT system handle

Provides access to information on the current connection to the Web server.

Note: Applies to SpeedScript programming, not ABL.

Syntax

```
WEB-CONTEXT [ :attribute | :method ]
```

attribute

An attribute of the WEB-CONTEXT handle.

method

A method of the WEB-CONTEXT handle.

Attributes

AUTO-DELETE-XML attribute	CONFIG-NAME attribute	CURRENT-ENVIRONMENT attribute
EXCLUSIVE-ID attribute	FORM-INPUT attribute	FORM-LONG-INPUT attribute
HANDLE attribute	HTML-CHARSET attribute	HTML-END-OF-LINE attribute
HTML-END-OF-PAGE attribute	HTML-FRAME-BEGIN attribute	HTML-FRAME-END attribute
HTML-HEADER-BEGIN attribute	HTML-HEADER-END attribute	HTML-TITLE-BEGIN attribute
HTML-TITLE-END attribute	INSTANTIATING-PROCEDURE attribute	IS-JSON attribute
IS-XML attribute	SESSION-END attribute	TYPE attribute
VALIDATE-XML attribute	X-DOCUMENT attribute	XML-ENTITY-EXPANSION-LIMIT attribute
XML-SCHEMA-PATH attribute	XML-STRICT-ENTITY-RESOLUTION attribute	XML-SUPPRESS-NAMESPACE-PROCESSING attribute

Methods

GET-BINARY-DATA( ) method	GET-CGI-LIST( ) method

GET-CGI-LONG-VALUE( ) method	GET-CGI-VALUE( ) method
GET-CONFIG-VALUE( ) method	INCREMENT-EXCLUSIVE-ID( ) method
URL-DECODE( ) method	URL-ENCODE( ) method

X-document object handle

A handle to an X-document object. You create the handle and assign it to a handle variable with the CREATE X-DOCUMENT statement.

Syntax

```
x-document-handle [ :attribute | :method ]
```

x-document-handle

A handle variable that references an X-document object.

attribute

An attribute of the X-document object.

method

A method of the X-document object.

Attributes

ENCODING attribute	ENTITY-EXPANSION-LIMIT attribute	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	NAME attribute	NAMESPACE-PREFIX attribute
NAMESPACE-URI attribute	NONAMESPACE-SCHEMA-LOCATION attribute	NUM-CHILDREN attribute
PUBLIC-ID attribute	SCHEMA-LOCATION attribute	SCHEMA-PATH attribute
STRICT-ENTITY-RESOLUTION attribute	SUBTYPE attribute	SUPPRESS-NAMESPACE-PROCESSING attribute
SYSTEM-ID attribute	TYPE attribute	UNIQUE-ID attribute

Methods

ADD-SCHEMA-LOCATION( ) method	APPEND-CHILD( ) method

CREATE-NODE( ) method	CREATE-NODE-NAMESPACE( ) method
GET-CHILD( ) method	GET-DOCUMENT-ELEMENT( ) method
IMPORT-NODE( ) method	INITIALIZE-DOCUMENT-TYPE( ) method
INSERT-BEFORE( ) method	LOAD( ) method (Handle)
REMOVE-CHILD( ) method	REPLACE-CHILD( ) method
SAVE( ) method	–

See also

CREATE X-DOCUMENT statement, X-noderef object handle

X-noderef object handle

A handle to a reference to an XML node. The X-noderef object is an ABL object that is a reference to any arbitrary node in an XML tree except a document node. You create the handle and assign it to a handle variable with the CREATE X-NODEREF statement.

An X-noderef object is not in a fully usable state until it has been associated with an X-document. Before this, you can only access its HANDLE, TYPE, PRIVATE-DATE, and UNIQUE-ID attributes without error. To avoid run-time errors, the following code shows one way to test the status of an X-noderef object:

```
cType = X-noderef-handle:SUBTYPE NO-ERROR.
IF VALID-HANDLE(X-noderef-handle) AND cType = 'ELEMENT' THEN DO:
   /*** your code ***/
END.
```

Syntax

```
x-noderef-handle [ :attribute | :method ]
```

x-noderef-handle

A handle variable that references an X-noderef object. You can use this handle as a parameter or return-value for attributes and methods that provide access to the underlying XML node.

attribute

An attribute of the X-noderef object.

method

A method of the X-noderef object.

Attributes

ATTRIBUTE-NAMES attribute	CHILD-NUM attribute	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	LOCAL-NAME attribute	NAME attribute
NAMESPACE-PREFIX attribute	NAMESPACE-URI attribute	NODE-VALUE attribute
NUM-CHILDREN attribute	OWNER-DOCUMENT attribute	SUBTYPE attribute
TYPE attribute	UNIQUE-ID attribute	–

Methods

APPEND-CHILD( ) method	CLONE-NODE( ) method
DELETE-NODE( ) method	GET-ATTRIBUTE( ) method
GET-ATTRIBUTE-NODE( ) method	GET-CHILD( ) method
GET-PARENT( ) method	INSERT-BEFORE( ) method
LONGCHAR-TO-NODE-VALUE( ) method	MEMPTR-TO-NODE-VALUE( ) method
NODE-VALUE-TO-LONGCHAR( ) method	NODE-VALUE-TO-MEMPTR( ) method
NORMALIZE( ) method	REMOVE-ATTRIBUTE( ) method
REMOVE-CHILD( ) method	REPLACE-CHILD( ) method
SET-ATTRIBUTE( ) method	SET-ATTRIBUTE-NODE( ) method

Notes

The value of the UNIQUE-ID attribute on this handle is guaranteed to be unique only for the life cycle of the single XML document instance to which it belongs (the currently-associated X-document object handle). Once the currently-associated x-document handle is deleted, any new XML document (referenced with a new X-document handle value) can re-use the UNIQUE-ID attribute values for the x-noderef objects of the deleted document. Thus, you can track the uniqueness of x-noderef object instances within an ABL session by remembering each pair of UNIQUE-ID attribute values from a given x-noderef object handle and its associated x-document object handle.

See also

CREATE X-NODEREF statement, X-document object handle

6

Handle Attributes and Methods Reference

This section contains reference entries that describe each handle attribute, handle method, COM object property, and COM object method that ABL supports. Handle attributes and methods, and COM object properties and methods, are all mechanisms that allow you to monitor and control the behavior of ABL handle-based objects (including widgets), system handles, and COM objects.

For information on ABL widgets, see the Widget Reference. For information on all other handle-base objects and system handles, see the Handle Reference. For information on handle attributes and methods that apply to SpeedScript, see the Handle Reference.

Each handle-based object has a set of attributes and methods (referred to generally as *handle attributes* and *handle methods*). Each COM object also has a set of properties and methods. This section describes every handle attribute and method available in ABL, but describes only the COM object properties and methods that directly support the ActiveX control container technology in ABL. All other Automation objects and ActiveX controls that you access from ABL provide their own COM object properties and methods. For more information on these properties and methods, see the documentation that comes with each COM object.

In this section, names of handle attributes and methods appear in all-uppercase, while names of COM object properties and methods, which follow Visual Basic coding conventions, appear in mixed case.

This section begins by explaining the syntax for handle-based object references. The basic syntax is similar for both handle-based and COM objects. However, it has been extended for COM objects to support the unique features of Automation objects and ActiveX controls. This extended syntax for COM object references follows in a separate section.

The descriptions that follow refer to both compile-time and run-time behavior, features that the language generally supports and determines at compile time and actions directed by using these features at run time. When describing compile-time features or actions, this section references ABL or the ABL compiler. When describing ABL-directed actions taken at run time, this section references the ABL Virtual Machine (AVM).

Note:

In character interfaces, all handle attributes and methods that reference pixels (for example, the HEIGHT-PIXELS attribute and the GET-TEXT-HEIGHT-PIXELS method) use a system default pixel value for the equivalent value in characters.

Properties and methods of classes are conceptually analogous to but different from either handle attributes and methods or COM object properties and methods. Class-based properties and methods participate in class hierarchies that you typically define, extend, and instantiate as objects of a user-defined class or interface type. All handle attributes and methods are associated with ABL handle-based objects, and are entirely defined and made available by ABL. COM object properties and methods are associated with pre-defined COM objects (Automation objects and ActiveX controls). The syntax for accessing all types of attributes, methods, and properties is very similar, but differ depending on the type of object with which they are associated. This section describes how to access the attributes, properties, and methods of respective ABL handle-based objects and COM objects only. For more information on accessing properties and methods of a class-based object, see the Class-based property access and Class-based method call reference entries.

Accessing handle attributes and methods

A *handle attribute* is a value that defines a visible, functional, and other characteristic of an ABL handle-based object. System handles also have attributes that describe and control certain handle-based object or system states. Attributes can be readable, writeable, or both. Readable means that your code can assign the value of the attribute to a variable or reference its value in an expression. Writeable means that your code can change the value of an attribute and thereby change the associated characteristic of the object or system state. Whether or not an attribute is readable or writeable depends on a number of factors (for example, the widget type, system handle type, widget realization, etc.).

A *handle method* is a specialized function associated with an ABL handle-based object that performs an action on the object, or alters the behavior of the object. Some system handles also have methods that affect certain widget and system behaviors. All methods return a value and some methods require parameters. The return value usually is a logical value specifying whether or not the execution of the method was successful. However, some methods return other types of information.

Handle-based references

You must access all handle attributes and methods with reference to a handle-based object or its handle using this general syntax:

Syntax

```
handle-based-reference : attribute-or-method-access
  [ IN container-widget-name ]
```

The *handle-based-reference* must conform to the following syntax:

```
{ object-name-reference | handle-reference }
```

An *object-name-reference* is a name reference to a static handle-based object, such as the defined name of a frame or window, or of another static handle-based object, such as a temp-table or buffer. For more information, see Object name references. A *handle-reference* can be a handle value that references any handle-based object (static or dynamic), or it can be a system handle. For more information, see Handle references.

The *attribute-or-method-access* identifies a given attribute or method associated with the specified *handle-based-reference*.

The *container-widget-name* is a name reference to a static container widget for the static widget referenced by *object-name-reference*. You need it only if the static widget reference is ambiguous.

For more information on *attribute-or-method-access* and how to access:

- Attributes, see Handle attribute access
- Methods, see Handle method calls

The following sections provide more information on the *object-name-reference* and *handle-reference* options.

Object name references

An *object-name-reference* can have the following syntax, depending on the static handle-based object:

Syntax

```
[ object-type ] object-name
```

The *object-type* is the type of the static handle-based object specified in the DEFINE statement for the handle-based object, such as FRAME in the DEFINE FRAME statement or TEMP-TABLE in the DEFINE TEMP-TABLE statement. For a data representation widget associated with a field or variable, *object-type* is the widget specified using the VIEW-AS phrase, such as COMBO-BOX for VIEW-AS COMBO-BOX.

The *object-name* is the name that you specify for the static handle-based object using the DEFINE statement for the handle-based object. For a data representation widget, it is the name of the field or variable associated with the widget.

For static frame fields, you only need to provide the *object-name* (field or variable name) to reference the associated widget, and ABL looks for its data representation widget in a frame. However, for most other static handle-based objects, you must qualify the *object-name* with the *object-type*, for example, BUFFER Customer, QUERY qCust, or TEMP-TABLE ttCust.

For example, in this code fragment, the handle to a frame field (`dDate`) and the handle to a buffer object (`bCust`) are both returned and assigned to handle variables using the HANDLE attribute of each handle-based object:

```
DEFINE VARIABLE hField  AS HANDLE NO-UNDO.
DEFINE VARIABLE hObject AS HANDLE NO-UNDO.
DEFINE VARIABLE dDate   AS DATE   NO-UNDO VIEW-AS FILL-IN.

DEFINE BUFFER bCust FOR Customer.

DEFINE FRAME AFrame
  dDate  bCust.Name.

hField  = dDate:HANDLE /* Widget name found in frame AFrame */.
hObject = BUFFER bCust:HANDLE.
```

Handle references

A *handle-reference* allows you to directly reference a handle-based object using a handle value. You can set this handle value by assigning the HANDLE attribute of a static handle-based object (see Object name references), by creating a dynamic handle-based object, by assigning the handle value provided by a system handle, or from any other source of handle values, such as method or function return values. Examples of handle references include CURRENT-WINDOW and BUFFER bCust:HANDLE (from the previous example).

You can access the handle to a static handle-based object using a HANDLE data type (field or variable) that you set to the value of the HANDLE attribute on the static object name reference.

You can access the handle for a dynamic handle-based object through a HANDLE data type (field or variable) whose value is initially set using the CREATE statement for the specified handle-based object. You can also access the handle value to some dynamic handle-based objects, as in the case of a procedure object, that you initially obtain using an appropriate RUN statement or system handle.

The HANDLE attribute of a given object handle can also provide a valid handle value that references the same handle-based object as the given object handle, itself. For example, the following assignment statements set both hProc1 and hProc2 to the same running procedure object using the THIS-PROCEDURE system handle and its HANDLE attribute, and the MESSAGE statement displays the FILE-NAME attribute for the same procedure object:

```
DEFINE VARIABLE hProc1 AS HANDLE NO-UNDO.
DEFINE VARIABLE hProc2 AS HANDLE NO-UNDO.

/* hProc1 equals hProc2 */
hProc1 = THIS-PROCEDURE.
hProc2 = THIS-PROCEDURE:HANDLE.

MESSAGE hProc1:FILE-NAME SKIP hProc2:FILE-NAME VIEW-AS ALERT-BOX.
```

Handle attribute access

To access a handle attribute, use the following syntax:

Syntax

```
widget-reference : attribute-name [ IN container-widget-name ]
```

The *widget-reference* is a reference to a handle-based object (see Handle-based references). The *attribute-name* specifies the ABL-defined name of a handle-based object attribute. The *container-widget-name* is a name reference to a static container for a static widget. You need it only if the static widget reference is ambiguous without it, as shown in the following example. For more information on accessing attributes, see Accessing handle attributes and methods.

To read an attribute value, assign the attribute value to a field or variable of a compatible data type, pass the attribute as an input parameter to some procedure, function, or method, or include the attribute in an expression. To write an attribute value, assign the value to the attribute or pass the attribute as an output parameter to some procedure, function, or method.

The following example repositions a selection list (Select-1) to another row in the specified frame (SelectFrameA):

```
DEFINE VARIABLE Select-1 AS CHARACTER NO-UNDO
  VIEW-AS SELECTION-LIST SIZE 50.0 BY 10.0.

DEFINE FRAME SelectFrameA Select-1.
DEFINE FRAME SelectFrameB Select-1.

Select-1:ROW IN FRAME SelectFrameA = Select-1:ROW + 2.
```

Note that ABL assumes that the second reference to Select-1 within the same assignment statement is in the same frame that is specified for the first reference.

Handle method calls

To invoke (execute) a handle method, use the following syntax:

Syntax

```
widget-reference : method-name ( [ parameter-list ] )
  [ IN container-widget-name ]
```

The *widget-reference* is a reference to a handle-based object (see Handle-based references). The *method-name* specifies the ABL-defined name of a handle method, and *parameter-list* specifies any parameter list for the method. The *container-widget-name* is a name reference to a static container for a static widget. You need it only if the static widget reference is ambiguous, as shown in the following example. To execute a handle method, you can assign the return value directly to a variable, include the method in an expression, or directly invoke the method as a statement (terminated with a period), ignoring its return value. For more information on calling handle methods, see Accessing handle attributes and methods.

The following example executes the ADD-FIRST( ) method for a selection list (Select-1) in two different ways—assigning the return value to a logical variable (methRtn) and invoking it directly, without assigning a value:

```
DEFINE VARIABLE Select-1 AS CHARACTER NO-UNDO
  VIEW-AS SELECTION-LIST SIZE 50.0 BY 10.0.
DEFINE VARIABLE methRtn  AS LOGICAL   NO-UNDO.

DEFINE FRAME SelectFrameA Select-1.
DEFINE FRAME SelectFrameB Select-1.

methRtn = Select-1:ADD-FIRST("BLUE") IN FRAME SelectFrameA.
Select-1:ADD-FIRST("GREEN") IN FRAME SelectFrameB.
```

Chained handle references

A chained handle reference has the following syntax:

Syntax

```
widget-reference : attribute-or-method-reference
  [ : attribute-or-method-reference  ] ...
```

A chained handle reference allows you to reference multiple object handles in a series that terminates with a final handle attribute or method. The *widget-reference* specifies a handle-based object (see Handle-based references). The *attribute-or-method-reference* can be any appropriate attribute or method of the referenced handle-based object immediately preceding it. When you use this syntax, the middle handle attributes and methods must return a data type of HANDLE in order to allow an additional handle reference in the chain, until the final handle attribute or method is accessed, which can return any ABL data type.

The following example shows a chained handle reference (see also Widget phrase, HANDLE function):

```
DEFINE VARIABLE hBuff AS HANDLE NO-UNDO.

CREATE BUFFER hBuff FOR TABLE "Customer".
MESSAGE hBuff:BUFFER-FIELD(3):NAME.
```

Widget color, font, and measurement values

Some entries in this section refer to color and font values (for example, BGCOLOR and FONT). These values are color and font numbers established for your system. Some entries in this section also refer to character units, an ABL unit of measure for specifying portable widget sizes and positions in graphical environments. For more information on color and font values, or character units and their relationship to pixels, see *OpenEdge Programming Interfaces*.

Note: When you assign a decimal value to an attribute representing a measurement in character units, ABL automatically rounds the assigned value to the nearest decimal value that corresponds to whole pixel units.

Accessing COM object properties and methods

A *COM object property* is a value that defines the visible, functional, and other characteristics of a COM object (ActiveX Automation object or ActiveX control). An ActiveX control property is classified as a design-time or run-time property depending on when you can change it. A design-time property can be changed using the Properties Window of the AppBuilder. A run-time property can be changed from ABL at run time. Generally, you can read both design-time and run-time properties at run time. In all other respects, COM object properties are functionally analogous to handle attributes.

A *COM object method* is a specialized function associated with a COM object that performs an action on the COM object or alters the behavior of the COM object. COM object methods may or may not return a value and may or may not require parameters. A return value may be a component handle to another COM object; however, many methods return other types of information or no information at all. Like handle methods, you execute COM object methods by calling them directly as statements rather than as part of an expression. In all other respects, COM object methods are functionally analogous to handle methods.

The basic syntax for accessing COM object properties and methods from ABL is similar to accessing handle attributes and methods. These are the main differences:

- You must access a COM object property or method with reference to an instantiated COM object. ABL supports the COM-HANDLE data type to provide COM object references for accessing properties and methods.
- You might have to specify the parameters of COM object methods with more type information, depending on the method and how the COM object is implemented.
- All COM objects are dynamic objects, so you never qualify a COM object reference by a static container reference (such as a static frame or menu widget).

Syntax for accessing COM object properties and methods

This section describes the three basic types of ABL statements for accessing properties and methods.

- Property write
- Property read
- Method call

Note: You can access COM object properties and methods from many other types of ABL statements that use these properties and methods in a similar fashion. The following statements, however, illustrate all the basic mechanisms for accessing COM objects from ABL.

Property write

The following syntax describes a basic assignment (=) statement for writing a COM object property:

Syntax

```
Com-Handle-Var :COMProperty [ AS Data-type ] = expression
  [ NO-ERROR ]
```

Com-Handle-Var is any COM-HANDLE variable set to the handle of an instantiated COM object. *COMProperty* is a COM object property access (see COMProperty). *Data-type* is any supported COM data type (see Data-type) that is compatible with *expression*. The *expression* is any combination of ABL elements that results in a single value. NO-ERROR allows the application to handle any ABL ERROR conditions raised by the statement.

Property read

The following syntax describes a basic assignment (=) statement for writing a COM object property:

Syntax

```
[ { field | COMProperty } = ]
  Com-Handle-Var :COMProperty [ NO-ERROR ]
```

A *field* is any ABL variable, database field, class-based data member or property, or handle attribute of a compatible data type. *COMProperty* is a COM object property access (see COMProperty). *Com-Handle-Var* is any COM-HANDLE variable set to the handle of an instantiated COM object. NO-ERROR allows the application to handle any ABL ERROR conditions raised by the statement.

Method call

The following syntax describes either a basic assignment (=) statement for invoking a COM object method that returns a value or the basic statement for invoking a COM object method directly that ignores any returned value:

Syntax

```
{ field } =
  Com-Handle-Var:Method-Name( [ param [ , param ] ... ] )
  [ NO-ERROR ]
```

```
Com-Handle-Var:Method-Name( [ param [ , param ] ... ] )
  [ NO-ERROR ]
```

field

Any ABL variable, database field, class-based data member or property, or handle attribute of a compatible data type. This can also be a COM object property (see COMProperty).

Com-Handle-Var

Any COM-HANDLE variable set to the handle of an instantiated COM object.

Method-Name

The name of the method to invoke.

param

The following syntax describes how to specify a parameter for a COM object method:

```
{ [ OUTPUT | INPUT-OUTPUT ] expression [ AS Data-type ]
  [ BY-POINTER | BY-VARIANT-POINTER ] }
```

expression

Can be any valid ABL expression or data element.

Data-type

A supported COM data type that is compatible with the *expression*. For information on COM data types and on OUTPUT, INPUT-OUTPUT, BY-POINTER, and BY-VARIANT-POINTER, see Data-type.

There is currently no support for named parameters, for example: *Method-Name*(Color="GREEN", Shape="SQUARE").

If a parameter is optional you may omit it from the parameter list, but you still need to retain the comma to indicate its position. For example, ComObject:ComMethod(param1,param2,,,param5).

NO-ERROR

Allows the application to handle any error condition raised by the statement.

Note: You can optionally chain COM object references.

COMProperty

The following syntax describes a COM object property access with optional chaining of COM object references:

Syntax

```
[ { COMProperty | COMMethod } : ] ...
  Property-Name [ ( index [ , index ] ... ) ]
```

Property-Name is the name of the accessed property. The optional multi-level *index* is an integer expression as required by the property. You must not follow the colon separator by a space.

Any *COMProperty* or *COMMethod* that precedes the final *Property-Name* must return a COM object reference that provides access to the immediately following *Property-Name*. The final property access can return any supported COM data type.

Data-type

The following syntax lists the supported COM data types for specifying any *Data-type* required for accessing a COM object property or method:

Syntax

```
SHORT | FLOAT | CURRENCY | UNSIGNED-BYTE | ERROR-CODE | IUNKNOWN
```

The requirements for using the OUTPUT, INPUT-OUTPUT, BY-VARIANT-POINTER, BY-POINTER, and AS *Data-type* options depend on the COM object method or property, the implementation of the COM object, and how you plan to use the parameter or property in your application. In many cases, *expression* is all that you require for a property write or method parameter. For more information on and examples of accessing COM object properties and methods, see the information on COM object references in the topics on ActiveX Automation and in the topics on control container support in *OpenEdge Programming Interfaces*.

ACCELERATOR attribute

The key label of the keyboard accelerator for the menu item.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: MENU-ITEM widget

ACCEPT-CHANGES( ) method

Accepts changes to the data in one temp-table or all temp-tables in a ProDataSet object.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle

Syntax

```
ACCEPT-CHANGES( )
```

When you accept changes on a ProDataSet object handle, the AVM makes the rows in all after-image tables the current version of those rows, and empties the before-image tables. When you accept changes for a Buffer object handle, the AVM makes the rows in the after-image table the current version of those rows, and empties the before-image table. In either case, the AVM sets the BEFORE-ROWID attribute of every row in the after-image tables to the Unknown value (?), and the ROW-STATE of every row in the after-image tables to ROW-UNMODIFIED (0).

Note: When executing the method, there is no guarantee as to the contents of the buffers involved after the method finishes executing.

ACCEPT-ROW-CHANGES( ) method

Accepts changes to the data in one row of a ProDataSet temp-table.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
ACCEPT-ROW-CHANGES( )
```

When you accept changes for a temp-table row, the AVM makes the row in the after-image table the current version of the row, and then removes the before-image table row. The AVM also sets the BEFORE-ROWID attribute of the row in the after-image table to the Unknown value (?), and the ROW-STATE of the row in the after-image table to ROW-UNMODIFIED (0).

ACTIVE attribute

Indicates whether an individual data-relation between two ProDataSet object buffers is active or inactive. Set to TRUE to activate an individual data-relation. Set to FALSE to deactivate a data-relation.

Alternatively, you can activate or deactivate all data-relations in a ProDataSet object by setting the RELATIONS-ACTIVE attribute on the ProDataSet object handle. All data-relations in a ProDataSet object are active by default.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Data-relation object handle

Deactivate an individual data-relation in a ProDataSet object when you want a FILL operation to load data into a ProDataSet member buffer using the individual buffer's query. Likewise, you can reactivate a data-relation in a ProDataSet object after completing a FILL operation to use the data-relation for traversing the data after the data is loaded.

When the AVM encounters an inactive relation (or the last child buffer in the relation tree), during a FILL operation on a ProDataSet buffer object handle, the AVM does not fill the child buffers of that relation. When the AVM encounters an inactive relation during a FILL operation on a ProDataSet object handle, it treats the first child buffer of the inactive relation as a top-level table (including all rows from its data source) and fills each child buffer based on the data relation's query. If you do not want the AVM to treat the first child buffer of the inactive relation as a top-level table, set the FILL-MODE of that buffer to NO-FILL. The AVM does not fill any of the child buffers.

If the AVM encounters an inactive relation while navigating a ProDataSet object, it does not prepare or open a dynamic query for the child table, even if there is a browse associated with the relation's query. If you want to access the child temp-table, you must do so through a separate query, a FOR EACH statement, or some other standard ABL construct in your application code.

When you reactivate data-relations, the AVM does not automatically resynchronize the hierarchy of queries on buffers below the newly active relation. If you want to resynchronize the related buffers, use the SYNCHRONIZE( ) method on the parent buffer.

ACTOR attribute

Returns the value of the actor attribute for the SOAP-header-entryref object as a URL. Identifies the recipient of a header element.

Data type: CHARACTER

Access: Read-only

Applies to: SOAP-header-entryref object handle

If the SOAP-header-entryref object does not contain an actor attribute, this attribute returns the empty string.

Note: In SOAP Version 1.2, the ROLE attribute replaces the SOAP Version 1.1 ACTOR attribute. The AVM automatically assigns the value of the ACTOR attribute to the ROLE attribute when accessing a SOAP Version 1.2 Web service.

ADD-BUFFER( ) method

Adds one new buffer to a query object or dynamic ProDataSet object, without affecting the other buffers, if any.

Use the SET-BUFFERS( ) method to remove all prior buffers and set all buffers for the object at the same time.

Return type: LOGICAL

Applies to: ProDataSet object handle, Query object handle

Syntax

```
ADD-BUFFER ( buffer )
```

buffer

A handle to a buffer, or a CHARACTER expression that evaluates to the name of a **DEFINED** buffer (you cannot specify the default buffer), that the AVM searches for at run time.

Note: The maximum number of buffers for a query or ProDataSet is 200.

Note: Searching for a buffer using a handle is more efficient than a character expression. The AVM resolves a character expression at run time by searching in the current routine for a static buffer with that name. If not found, the AVM searches the enclosing main procedure. If still not found, the AVM searches up through the calling programs of the current routine, and their main procedures. Since a handle uniquely identifies the buffer, no such search is required.

The following is an example when the parameter is a handle to a buffer:

```
my-query-handle:ADD-BUFFER(BUFFER Customer:HANDLE).
```

The following is an example when the parameter is a CHARACTER expression:

```
DEFINE BUFFER bCustomer FOR Customer.
my-query-handle:ADD-BUFFER("bCustomer").
```

ADD-CALC-COLUMN( ) method

Creates a browse column from the specified properties and returns the handle of the new column. This method can be used only after the browse's query attribute has been set.

Return type: HANDLE

Applies to: BROWSE widget

Syntax

```
ADD-CALC-COLUMN ( datatype-exp , format-exp , initial-value-exp , label-exp
  [ , pos [ , widget-type ] ] )
```

datatype-exp

Character expression specifying the data type. Valid values are "CHARACTER", "DATE", "DECIMAL", "INTEGER", "INT64", or "LOGICAL".

format-exp

Character expression specifying the column's format.

initial-value-exp

Character expression specifying the initial value. This may be a null string.

label-exp

Character expression specifying the column's label.

pos

The optional integer value position of the browse column. If *pos* = 2, the column is the second column. If the position is not specified or the position is invalid, the new column is added at the end of the columns.

widget-type

An optional character expression specifying the widget type of the column. Valid values are "COMBO-BOX", "DROP-DOWN", "DROP-DOWN-LIST", "FILL-IN" and "TOGGLE-BOX". The default value is "FILL-IN".

If *datatype-exp* is not "LOGICAL", specifying "TOGGLE-BOX" as the widget type generates a run-time error.

The `widget-type` parameter is ignored in character mode.

The following is an example of adding a column in the browse's fifth position using this method:

```
CalcHnd5 = BrwsHndl:ADD-CALC-COLUMN("char", "AAA-99999", "ORD-37854",        "OrdNum",
 5).
```

The ADD-CALC-COLUMN( ) method may be used on a static browse as well as on a dynamic browse.

If the browse is already displayed, the REFRESH( ) method should be applied to the browse after columns are added using ADD-CALC-COLUMN( ). This will initially populate the viewport for the calculated column. The ROW-DISPLAY trigger would normally populate the column, but when ADD-CALC-COLUMN is being executed, the ABL calc-column handle is not yet set and, thus, cannot initially populate it.

ADD-COLUMNS-FROM( ) method

Creates a browse column for each field found in the specified buffer or table. If a field is found that already has a corresponding browse column created, it is ignored. This method can be used only after the browse's query attribute has been set.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
ADD-COLUMNS-FROM ( buffer-handle | table-name-exp [ , except-list ] )
```

buffer-handle

The handle of a buffer associated with the browse's query.

table-name-exp

The name of a table associated with the browse's query.

except-list

An expression that evaluates to a comma-separated list of field names to be excluded from the browse. Do not include the table name with these field names, and do not include spaces between field names.

The following is an example of adding browse columns from the Invoice table, excluding the fields, Amount and Total-Paid:

```
DEFINE VARIABLE ExcList AS CHARACTER INITIAL "Amount,Total-Paid".
. . .
AddCol = BrwsHndl:ADD-COLUMNS-FROM("invoice", ExcList).
```

The ADD-COLUMNS-FROM( ) method may be used on a static browse as well as a dynamic browse. When used on a static browse, the browse will become a NO-ASSIGN browse (you must make the database updates).

A dynamic browse column's validation expression is restricted. It may not contain a CAN-FIND function. To reference the field, the FRAME-VALUE function must be used.

To create a combo-box or toggle-box browse column for a buffer-field, specify the widget type in the buffer-field's VIEW-AS attribute. A buffer-field's VIEW-AS attribute setting remains in effect for subsequent columns created for that buffer-field, until you change the attribute setting, and changing the attribute setting does not affect previously created columns. (The VIEW-AS attribute has no effect in character mode.)

ADD-EVENTS-PROCEDURE( ) method
(Windows only; Graphical interfaces only)

Adds an external procedure to the list that the AVM searches for event procedures to handle ActiveX control events.

Return type: LOGICAL

Applies to: CONTROL-FRAME widget

Syntax

```
ADD-EVENTS-PROCEDURE ( procedure-handle )
```

procedure-handle

A handle to a persistent procedure or an otherwise active procedure on the call stack.

By default, the AVM searches the external procedure that created the current control-frame for the event procedure to handle an ActiveX control event. This method allows you to specify alternative procedure (.p and .w) files to search for the event handler.

When the AVM receives an ActiveX event, it searches for the event handler in order of the most recent procedure added to the search list and ends the search with the external procedure that created the control-frame. You can override an existing procedure by adding a different one to the search list. The AVM always uses the event handler in the most recently added procedure.

If the method succeeds in adding the procedure to the list, it returns TRUE. Otherwise, it returns FALSE.

ADD-FIELDS-FROM( ) method

Copies the field definitions from the specified source table to a temp-table. It is intended for use when a temp-table represents a join. If it finds fields that are already in the temp-table, it ignores them.

This method cannot be called after TEMP-TABLE-PREPARE( ) has been called unless CLEAR( ) is called first.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
ADD-FIELDS-FROM ( { source-buffer-handle-exp | source-table-name-exp }
[ , except-list-exp] )
```

source-buffer-handle-exp

An expression that evaluates to a buffer handle from which to copy the field definitions.

source-table-name-exp

An expression that evaluates to a table name from which to copy the field definitions.

except-list-exp

A character expression that evaluates to a comma-separated list of field names to be excluded from the new table definition.

The following example fragment creates a join temp-table from the Customer and Order tables:

```
DEFINE VARIABLE tth AS HANDLE NO-UNDO.

CREATE TEMP-TABLE tth.
tth:ADD-FIELDS-FROM("Customer").
tth:ADD-FIELDS-FROM("Order").
tth:TEMP-TABLE-PREPARE("cust-ord").
. . .
```

The following fragment creates a temp-table from the Customer table except for the `SalesRep` field:

```
tth:ADD-FIELDS-FROM("Customer", "SalesRep").
```

Note: There is a limit to the number of fields that can be accommodated in a temp-table object. The limit depends on how large the field information (initial value, validate information, help messages, etc.) is, but you should plan on a limit of approximately 1000 fields.

This method does not create any indexes. Either indexes must be added specifically through one of the ADD-INDEX methods, or a default index is created.

ADD-FIRST( ) method

Adds one or more items to the top of a combo box or selection list.

Return type: LOGICAL

Applies to: BROWSE widget (column), COMBO-BOX widget, SELECTION-LIST widget

Syntax

```
ADD-FIRST ( { item-list | label , value } )
```

item-list

A character-string expression that represents one or more items, delimiter-separated.

label

A character-string expression that represents the label of a label-value pair.

value

The value the AVM assigns to the field or variable if the user selects the corresponding label.

The delimiter is the value of the DELIMITER attribute, which defaults to comma. If the SORT attribute is TRUE, the AVM sorts new items by label before adding them to the widget. If the operation is successful, ADD-FIRST returns TRUE.

If the widget's entries consist of single items, use *item-list*. If the widget's entries consist of label-value pairs, use *label* and *value*.

If the widget's entries consist of single items, each call to ADD-FIRST can add multiple entries. If the widget's entries consist of label-value pairs, each call to ADD-FIRST can add one entry.

For browses, this method applies only to combo-box browse columns.

The following examples modify widgets whose entries consist of single items:

```
return-code = my-widget-hdl:ADD-FIRST("Seoul").
```

```
return-code = my-widget-hdl:ADD-FIRST("Bogota, Seoul, Los Angeles").
```

The following example modifies a combo-box widget of type INTEGER whose entries consist of label-value pairs:

```
return-code = my-widget-hdl:ADD-FIRST("Bogota", 15).
```

ADD-HEADER-ENTRY( ) method

Creates a new entry in a SOAP-header object's list of entries and associates the new entry with an existing SOAP-header-entry-ref object. The method returns the handle to the new entry.

Return type: LOGICAL

Applies to: SOAP-header object handle

Syntax

```
ADD-HEADER-ENTRY ( header-entryref )
```

header-entryref

The handle to the new SOAP-header-entryref object.

Following is an example of adding a SOAP-header-entryref object to a SOAP-header object:

```
CREATE SOAP-HEADER hSOAPHeader.
CREATE SOAP-HEADER-ENTRYREF hshEntry.
hSOAPHeader:ADD-HEADER-ENTRY (hshEntry).
```

ADD-INDEX-FIELD( ) method

Adds the specified field to the specified index of a temp-table. It requires the named index to be added first.

This method cannot be called after TEMP-TABLE-PREPARE( ) has been called unless CLEAR( ) is called first.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
ADD-INDEX-FIELD ( index-name-exp , field-name-exp [ , mode-exp ] )
```

index-name-exp

A character expression that evaluates to the name of the index to which the field is being added.

field-name-exp

A character expression that evaluates to the name of the field to add to the index.

mode-exp

An expression that evaluates to desc if it is descending or asc if it is ascending. The default is asc.

The following example fragment adds to a temp-table a new unique primary index field with two components, the first ascending, the second descending:

```
tth:ADD-FIELDS-FROM("Customer", "SalesRep").
tth:ADD-NEW-INDEX("abidx", TRUE, TRUE).
tth:ADD-INDEX-FIELD("abidx", "abfield1").
tth:ADD-INDEX-FIELD("abidx", "abfield2", "desc").
...
```

ADD-LAST( ) method

Adds one or more items to the bottom of a combo box, radio set, or selection list.

Return type: LOGICAL

Applies to: BROWSE widget (column), COMBO-BOX widget, RADIO-SET widget, SELECTION-LIST widget

Syntax (combo-box, selection-list, or browse column)

This is the syntax for a combo-box, selection-list, or browse column.

```
ADD-LAST ( { item-list | label , value } )
```

item-list

A character-string expression that represents one or more items, delimiter-separated.

label

A character-string expression that represents the label of a label-value pair.

value

The value the AVM assigns to the field or variable if the user selects the corresponding label.

Note: If the widget's entries consist of single items, use *item-list*. If the widget's entries consist of label-value pairs, use *label* and *value*.

For combo boxes and selection lists, the delimiter is the value of the DELIMITER attribute, which is comma by default. Also, if the SORT attribute is TRUE, ADD-LAST sorts the new items by label before adding them to the widget.

For browses, this method applies only to combo-box browse columns.

Syntax (radio-set)

This is the syntax for a radio-set.

```
ADD-LAST ( label , value )
```

label

A character-string expression that represents the label of a label-value pair.

value

An integer expression that represents the value of a label-value pair. When the radio set appears, if the user selects *label*, the AVM assigns *value* to the corresponding field or variable.

For radio sets, if the AUTO-RESIZE attribute is TRUE; the size of the radio set changes. Otherwise, the radio set is clipped.

For all applicable widgets, if the operation is successful, ADD-LAST returns TRUE.

Note: If the widget's entries consist of single items, each call to ADD-LAST can add multiple entries. If the widget's entries consist of label-value pairs, each call to ADD-LAST can add one entry.

The following examples modify widgets whose entries consist of single items:

```
return-code = my-combo-box-hdl:ADD-LAST("Seoul").
```

```
return-code = my-sel-list-hdl:ADD-LAST("Bogota, Seoul, Los Angeles").
```

The following example modifies a combo-box widget of type INTEGER whose entries consist of label-value pairs:

```
return-code = my-widget-hdl:ADD-LAST("Bogota", 15).
```

ADD-LIKE-COLUMN( ) method

Creates a browse column from the specified field and returns its handle. This method can be used only after the browse's query attribute has been set.

Return type: HANDLE

Applies to: BROWSE widget

Syntax

```
ADD-LIKE-COLUMN ( field-name-exp | buffer-field-handle
  [ , pos  [ , widget-type ] ] )
```

field-name-exp

The name of a field in one of the buffers associated with the browse's query. If the query is a join, the name must be qualified with the buffer name or the name of the table associated with the buffer.

buffer-field-handle

The handle of a buffer-field from a buffer associated with the browse's query.

pos

The optional integer value position of the browse column. If *pos* = 2, the column is the second column. If the position is not specified or the position is invalid, the new column is added at the end of the columns.

widget-type

An optional character expression specifying the widget type of the column. Valid values are "COMBO-BOX", "DROP-DOWN", "DROP-DOWN-LIST", "FILL-IN", and "TOGGLE-BOX". The default value is "FILL-IN".

The `widget-type` parameter is ignored in character mode.

A widget type of "COMBO-BOX" or "DROP-DOWN-LIST" is valid only when the buffer-field's data type is CHARACTER, DATE, DECIMAL, INTEGER, INT64 or LOGICAL. A widget type of "DROP-DOWN" is valid only when the buffer-field's data type is CHARACTER. A widget type of "TOGGLE-BOX" is valid only when the buffer-field's data type is LOGICAL. If the widget type is not valid for the buffer-field's data type, the AVM generates a run-time error.

If not specified, the AVM creates the browse column based on the widget type specified in the buffer-field's VIEW-AS attribute. (The VIEW-AS attribute has no effect in character mode.)

The ADD-LIKE-COLUMN( ) method can be used on a static browse as well as a dynamic browse. When used on a static browse, the browse will become a NO-ASSIGN browse (you must make the database updates).

A dynamic browse column's validation expression is restricted. It may not contain a CAN-FIND function. To reference the field, the FRAME-VALUE function must be used.

The following is an example of adding the Customer number field to the browse:

```
ColHdl = BrwsHndl:ADD-LIKE-COLUMN("Customer.CustNum").
```

ADD-LIKE-FIELD( ) method

Adds a field, like the specified source field, to the temp-table.

This method cannot be called after TEMP-TABLE-PREPARE( ) has been called unless CLEAR( ) is called first.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
ADD-LIKE-FIELD ( field-name-exp ,
  source-buffer-field-handle-exp | source-db-field-name-exp )
```

field-name-exp

A character expression that evaluates to the name of the field to be created in the temp-table.

source-buffer-field-handle-exp

A character expression that evaluates to a buffer-field handle from which to copy the field.

source-db-field-name-exp

A character expression that evaluates to a database field name from which to copy the field. The table name must be qualified with the database name.

The following example fragments add a field to a temp-table, the first from a named source and the second from a buffer-field handle source:

```
tth:ADD-LIKE-FIELD("ordno","Order.OrderNum").
```

```
tth:ADD-LIKE-FIELD(bfh:NAME, bfh).
```

Note: There is a limit to the number of fields that can be accommodated in a temp-table object. The limit depends on how large the field information (initial value, validate information, help messages, etc.) is, but you should plan on a limit of approximately 1,000 fields.

ADD-LIKE-INDEX( ) method

Adds an index, like the specified source index, to the temp-table.

This method cannot be called after TEMP-TABLE-PREPARE( ) has been called unless CLEAR( ) is called first.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
ADD-LIKE-INDEX ( index-name-exp , source-index-name-exp
  { , source-buffer-handle-exp | source-db-table-name-exp } )
```

index-name-exp

A character expression that evaluates to the name of the index to which the source index is being copied.

source-index-name-exp

A character expression that evaluates to the name of the index in the source table that is being copied to the temp-table.

source-buffer-handle-exp

A character expression that evaluates to a buffer handle from which to copy the index.

source-db-table-name-exp

A character expression that evaluates to a database table name from which to copy the index.

The following example fragment adds a new index to a temp-table like the name index in the Customer table:

```
tth:ADD-LIKE-INDEX("abidx","name","Customer").
```

ADD-NEW-FIELD( ) method

Adds a field with the specified properties to the temp-table. Additional properties can be manipulated by creating a buffer-field object for this field.

This method cannot be called after TEMP-TABLE-PREPARE( ) has been called unless CLEAR( ) is called first.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
ADD-NEW-FIELD ( field-name-exp , datatype-exp [ , extent-exp [ , format-exp
  [ , initial-exp [ , label-exp [ , column-label-exp ] ] ] ] ] )
```

field-name-exp

A character expression that evaluates to the name of the field to be created in the temp-table.

datatype-exp

A character expression that evaluates to the data type of the specified field.

extent-exp

An integer expression specifying the extent of an array. If *extent-exp* is 0, 1 or the Unknown value (?), it is ignored.

format-exp

A character expression that evaluates to the data format for the defined data type. If *format-exp* is "" or the Unknown value (?), it is ignored and the default format of the specified data type is used.

initial-exp

An expression that evaluates to the initial value of the defined field. *initial-exp* can be any compatible data type, but is usually character. If *initial-exp* is not entered, the default for the data type is used.

label-exp

An optional character expression that evaluates to the label of the defined field. If you do not specify a value, or you pass the Unknown value (?), *label-exp* defaults to the value of the *field-name-exp* parameter.

column-label-exp

An optional character expression that evaluates to the label of the column associated with the defined field. If you do not specify a value, or you pass the Unknown value (?), *column-label-exp* defaults to the value of the *label-exp* parameter (or the *field-name-exp* parameter, if the *label-exp* parameter is not specified).

The following example fragment adds a new character field called "abfield" which is initialized to "abc" to a temp-table:

```
tth:ADD-NEW-FIELD("abfield","char",0,"X(3)","abc").
```

Note: There is a limit to the number of fields that can be accommodated in a temp-table object. The limit depends on how large the field information (initial value, validate information, help messages, etc.) is, but you should plan on a limit of approximately 1,000 fields.

ADD-NEW-INDEX( ) method

Adds a new empty index with the specified name to the temp-table. Index components must be added with the ADD-INDEX-FIELD( ) method.

This method cannot be called after TEMP-TABLE-PREPARE( ) has been called unless CLEAR( ) is called first.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
ADD-NEW-INDEX ( index-name-exp [ , unique-exp [ , primary-exp
  [ , wordix-exp ]]] )
```

index-name-exp

A character expression that evaluates to the name of the created index.

unique-exp

A logical expression that evaluates to TRUE if this index is unique.

primary-exp

A logical expression that evaluates to TRUE if this is the primary index.

wordix-exp

A logical expression that evaluates to TRUE if this is a word index.

The following example fragment adds to a temp-table a new unique primary index field with two components, the first ascending, the second descending:

```
tth:ADD-FIELDS-FROM("Customer", "SalesRep").
tth:ADD-NEW-INDEX("abidx", TRUE, TRUE).
tth:ADD-INDEX-FIELD("abidx", "abfield1").
tth:ADD-INDEX-FIELD("abidx", "abfield2", "desc").
...
```

ADD-PARENT-ID-RELATION( ) method

Adds a data-relation object for a pair of parent and child buffers that is based on the RECID of the parent buffer to a dynamic ProDataSet object. It also sets the PARENT-ID-RELATION attribute to TRUE.

Return type: HANDLE

Applies to: ProDataSet object handle

Syntax

```
ADD-PARENT-ID-RELATION ( parent-buffer-handle , child-buffer-handle ,
  [parent-id-field[ , parent-fields-before [ , parent-fields-after]]] )
```

parent-buffer-handle

Specifies a handle to the parent buffer in the data-relation object.

child-buffer-handle

Specifies a handle to the child buffer in the data-relation object.

parent-id-field

Indicates the RECID in *child-buffer-handle* on which this *parent-id-relation* object is based. This is an optional CHARACTER expression. The default value is the name of the *parent-buffer-handle* appended with "_id".

parent-fields-before

Indicates an optional CHARACTER expression evaluating to a comma-separated list of fields from the parent record to be written to XML before any nested *child-buffer-handle* records.

parent-fields-after

An optional CHARACTER expression evaluating to a comma-separated list of fields from the parent record to be written to XML after all nested *child-buffer-handle* records.

See also

PARENT-ID-RELATION attribute

ADD-RELATION( ) method

Adds a data-relation object for a pair of parent and child buffers to a dynamic ProDataSet object.

Return type: HANDLE

Applies to: ProDataSet object handle

Syntax

```
ADD-RELATION ( parent-buffer-handle , child-buffer-handle ,
  [ pairs-list [ , reposition-mode [ , nested [ , not-active [ , recursive
  [ , foreign-key-hidden  ]]]]]])
```

parent-buffer-handle

A handle to the parent buffer in the data-relation object.

child-buffer-handle

A handle to the child buffer in the data-relation object.

pairs-list

A character expression that evaluates to a comma-delimited list of parent-field, child-field pairs describing the relationship between parent and child buffers in the data-relation object using the following syntax:

```
"parent-field1,child-field1[,parent-fieldn,child-fieldn ]..."
```

The first field in the pair is from the parent buffer, the second field is from the child buffer. This list can contain no embedded spaces. When filling the ProDataSet object, the AVM retrieves data for the child buffer based on an equality match between all pairs of fields unless the data-delation is deactivated or there is an explicit query definition for the data source of the child buffer.

You can define a query for the data source of the child buffer, or you can supply custom logic in response to FILL events that take over complete responsibility for filling one level of the ProDataSet object. In these cases, the *pairs-list* is not used.

reposition-mode

The reposition mode of the relation between the parent and child temp-tables. If TRUE, the relation mode is REPOSITION. If FALSE, the relation mode is SELECTION. The default value is FALSE.

When the relation mode is SELECTION, the method fills the child temp-table of the data-relation object with all records related to the current parent. When the relation mode is REPOSITION, the relation is effectively ignored during a FILL, and the child of the relation is treated as if it were a top-level buffer.

When navigating a filled ProDataSet object with a SELECTION relation, related data is filtered as it is browsed. This means the child query of the relation is filtered to make available only children of the current parent, and the query is re-opened each time the parent table is repositioned. When navigating a filled ProDataSet object with a REPOSITION relation, the child table query is always set to match all the rows in the child table, and is not re-opened when the parent changes. Only the buffer for the child is repositioned to the matching child for the current parent.

nested

A LOGICAL expression where TRUE directs the AVM to nest child rows of ProDataSet buffers within their parent rows when writing the XML representation of data. This also causes the XML Schema definitions for the related temp-tables to be nested. When FALSE, all child rows are written after all parent rows are written. The default value is FALSE.

not-active

An optional LOGICAL expression where FALSE causes the data-relation to be inactive. This allows you to have two relations between the same two ProDataSet temp-table buffers, but only have one active relation at a time.

recursive

An optional LOGICAL expression where TRUE causes the data-relation to be filled recursively. This mode instructs the ProDataSet FILL to load self-referencing elements. That is, an element can reference a child element that already either directly or indirectly references the parent element in the hierarchy. Examples of this hierarchy are bill-of-materials parent/child tables or organization chart manager/employee relationship tables.

foreign-key-hidden

An optional LOGICAL expression where TRUE directs the AVM to omit foreign key fields in nested data-relations when writing the XML representation of data. This also causes the XML Schema definition for the ProDataSet to indicate that the foreign key fields are optional. Since the parent record in a nested relationship contains the foreign key, suppressing it in the nest child record is efficient. If this option is used, the NESTED argument must be TRUE.

When working with large ProDataSets, omitting foreign keys in nested child records can yield smaller XML documents, more efficient network transfers, and performance gains with the READ-XML( ) and WRITE-XML( ) methods.

Care must be taken when deciding to use this feature. The READ-XML( ) method automatically populates foreign keys in nested child records with the values in the outer parent record when the foreign key is omitted from the XML document. Unless you are sure that the receiver of the XML document will do the same, you should not use this option in your nested data-relations.

For example, while .NET can read this XML document and populate an ADO .NET DataSet, it will create rows in the child DataTable with a null value for the foreign key fields.

See also

FOREIGN-KEY-HIDDEN attribute, RECURSIVE attribute

ADD-SCHEMA-LOCATION( ) method

An XML Schema file location is specified by providing a pair of values: a namespace and a physical location. This method allows you to specify that value pair. The XML Schema file is used by an X-document or SAX-reader object to validate XML content.

Return type: LOGICAL

Applies to: X-document object handle, SAX-reader object handle

Syntax

```
ADD-SCHEMA-LOCATION ( targetNamespace, location )
```

targetNamespace

A CHARACTER expression evaluating to the target namespace of the schema, or an empty string ("") or the Unknown value (?) if the location doesn't contain a namespace.

location

A CHARACTER expression evaluating to the location of the XML Schema file. This value can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

Provides the location of an XML Schema file for the parser by specifying the namespace and physical location of the XML Schema file.

You can call this method more than once to create a list of schema locations.

Note that namespace and XML Schema file locations specified programatically with this method take precedence over namespaces or schemas declared in XML documents or imported elements.

This method and the SCHEMA-LOCATION attribute are both used for setting the XML Schema file location. This method is added for convenience.

ADD-SOURCE-BUFFER( ) method

Adds a database buffer to a dynamic data-source object at run time.

Return type: LOGICAL

Applies to: Data-source object handle

Syntax

```
ADD-SOURCE-BUFFER ( buffer-handle, key-fields )
```

buffer-handle

A handle to the database buffer you are adding.

key-fields

A character expression that evaluates to a comma-separated list of one or more database table fields that constitute a unique key that the AVM can use to find a record in the table.

ADD-SUPER-PROCEDURE( ) method

Associates a super procedure file with a procedure file or with the current ABL session. When a procedure file invokes an internal procedure or a user-defined function, ABL searches for it, among other places, in the super procedures (if any) of the procedure file and of the current ABL session. The procedure-search option determines which procedures are searched.

For more information on the rules that ABL uses to search for internal procedures and user-defined functions, see Search rules.

For a sample program that uses the ADD-SUPER-PROCEDURE method, see the reference entry for the RUN SUPER statement.

Returns FALSE for a Web service procedure.

Return type: LOGICAL

Applies to: Procedure object handle, SESSION system handle

Syntax

```
ADD-SUPER-PROCEDURE ( super-proc-hdl [ , proc-search ] )
```

super-proc-handle

The handle of a running persistent procedure that you want to make a super procedure of the local procedure or of the current ABL session.

ADD-SUPER-PROCEDURE returns FALSE if *super-proc-hdl* is not a valid handle, or if the AVM detects that the method was not successful. Otherwise, the method returns TRUE.

proc-search

Optional expression that determines which super procedures are searched when super-proc-hdl invokes RUN SUPER or the SUPER function. Valid values are SEARCH-SELF (or 1) or SEARCH-TARGET (or 2). The default, if there is no entry, is SEARCH-SELF. The search commences in the super procedure stack of *super-proc-hdl*.

Consider the following:

- SEARCH-SELF starts searching in the procedure file that initiated the current internal procedure or user-defined function.
- SEARCH-TARGET starts searching the super procedures of the procedure file that originally invoked the current internal procedure or user-defined function (the procedure with the original RUN statement). If the procedure was RUN . . . IN *procedure-handle*, SEARCH-TARGET searches the super procedures of *procedure-handle*.
- A given *super-proc-hdl* can be added as either SEARCH-TARGET or SEARCH-SELF, but cannot be added as both. If *proc-search* is set for a *super-proc-hdl*, then any attempt to change its value generates a run-time warning, but the ADD-SUPER-PROCEDURE( ) method succeeds. The warning message "Changing proc-search-string for procedure <.p-name> from <string> to <string>" is presented to indicate that the application is using an instance of a given super procedure in an inconsistent manner. This warning message can be suppressed by using the SESSION:SUPPRESS-WARNINGS-LIST attribute. In addition, the warning message can be avoided by creating two instances of *super-proc-hdl*, one identified as SEARCH-TARGET and the other identified as SEARCH-SELF.

Associating a super procedure with a procedure

The following example associates a super procedure with the current procedure:

```
THIS-PROCEDURE:ADD-SUPER-PROCEDURE(my-super-proc-hdl,SEARCH-SELF).
```

The following example:

- Associates a super procedure with a procedure that the current procedure is working for.
- Requests that the super procedure stack associated with local-proc-hdl be searched rather than the stack associated with my-super-proc-hdl when RUN SUPER is invoked in *super-proc-hdl*.

```
local-proc-hdl:ADD-SUPER-PROCEDURE(my-super-proc-hdl,SEARCH-TARGET).
```

The procedure to which you add a super procedure is called the *local procedure* of the super procedure.

Associating a super procedure with the current ABL session

The following example associates a super procedure with the current ABL session:

```
SESSION:ADD-SUPER-PROCEDURE(my-super-proc-hdl).
```

When you do this, the AVM automatically associates the super procedure with all the session's procedures—persistent and nonpersistent—without your having to change their code in any way. This technique lets you replace occurrences of the following:

```
THIS-PROCEDURE:ADD-SUPER-PROCEDURE(super-proc-hdl).
```

In individual procedures with a single occurrence of the following:

```
SESSION:ADD-SUPER-PROCEDURE(super-proc-hdl).
```

Super procedure stacking

You can associate multiple super procedures with a single local procedure or with the current ABL session. When you do this, the AVM stores (and later on, searches) the corresponding procedure handles in last in first out (LIFO) order—the handle of the most recently added super procedure first, the handle of the next most recently added super procedure second, etc.

A collection of super procedure handles associated with a local procedure or with the current ABL session is called a super procedure **stack**. The handle of the most recently added super procedure occupies the **top** of the stack.

If you add a super procedure that is already in the stack, the AVM removes the previous occurrence of the super procedure handle from the stack and adds the new occurrence to the top of the stack—all without reporting an error.

Super procedure chaining

You can add a super procedure to a super procedure. For example, imagine the following scenario:

1. A, B, and C are procedure files running persistently.
2. B is a super procedure of A.
3. C is a super procedure of B.

B is a super procedure (of A) and has a super procedure (C).

When you add a super procedure to a super procedure, the result is a super procedure **chain**, each **link** of which consists of two elements: a local procedure and its super procedure. When the AVM searches a super procedure chain, it does not proceed to the next link unless the current link's super procedure element explicitly invokes its super version (by using the RUN SUPER statement or the SUPER function).

For example, imagine the following scenario:

1. A, B, and C, and X are procedure files running persistently.
2. add-record is an internal procedure different versions of which reside in A, B, and C.
3. B is a super procedure of A.
4. C is a super procedure of B.
5. X says RUN add-record IN A.

The following events occur:

1. The AVM searches A for add-record and runs it if found.
2. If and only if A's add-record exists and says RUN SUPER, the AVM searches B for add-record and runs it if found.

   Note: If A does not contain add-record, the following events occur: If B contains add-record, the AVM runs it. If B does not contain add-record, the AVM does not search for add-record in C.

3. If and only if B's add-record exists and says RUN SUPER, the AVM searches C for add-record and runs it if found.

In this way, the AVM avoids excessive and possibly circular searching.

Search rules

The AVM searches for internal procedures and user-defined functions depending on how the internal procedure or user-defined function is invoked. The search rules illustrated in the first three cases assume that all the super procedures were added with no *proc-search* value or with a *proc-search* value of SEARCH-SELF. The fourth case illustrates the search process when a super procedure is added with a *proc-search* value of SEARCH-TARGET.

Case 1: When the AVM encounters a statement like the following:

```
RUN add-record('Customer').
```

The AVM searches for add-record as follows:

1. As an internal procedure in the local procedure
2. As an internal procedure in a super procedure of the local procedure
3. As an internal procedure in a super procedure of the ABL session
4. As an external procedure file `add-record.p` or `add-record.r`

Case 2: When the AVM encounters a statement like the following:

```
RUN add-record IN my-proc-hdl('Customer').
```

The AVM searches for add-record as follows:

1. As an internal procedure in my-proc-hdl
2. As an internal procedure in a super procedure of my-proc-hdl
3. As an internal procedure in a super procedure of the ABL session

Case 3: When the AVM encounters a statement like the following:

```
add-record('Customer').
```

The AVM searches for add-record as follows:

1. As a user-defined function in the local procedure
2. As a user-defined function in a super procedure of the local procedure
3. As a user-defined function in a super procedure of the ABL session

---

Note: The rules of Case 3 apply whether or not the user-defined function's declaration (function prototype) includes the IN *proc-hdl* option. In Case 3, *proc-hdl* represents the local procedure. For more information on function prototypes of user-defined functions, see *ABL Essentials*.

Search rules for SEARCH-TARGET

Case 4: A procedure, `main.p`, has added three super procedures, S1, S2, and S3 (in that order). Each of these super procedures has added its own super procedures, S1A, S1B, S2A, S2B, S3A, S3B. The procedure, add-record, exists in three places: in S1, in S2 where it contains a RUN SUPER statement, and in S2A.

When the AVM encounters a statement like "`RUN add-record('customer').`", it searches for the add-record procedure:

1. As an internal procedure in the local procedure, `main.p`
2. Then as an internal procedure in S3, and then in S2 where it is found

The following code for `main.p` shows the differences in the initial setup of this case, which compares the use of no *proc-search* value (Instance 1) with the use of a value of SEARCH-TARGET (Instance 2):

```
/* main.p */
DEFINE VARIABLE h AS HANDLE NO-UNDO.

RUN s1.p PERSISTENT SET h.
THIS-PROCEDURE:ADD-SUPER-PROC(h).

RUN s2.p PERSISTENT SET h.
/* Instance 1
THIS-PROCEDURE:ADD-SUPER-PROC(h). */
/* Instance 2 */
THIS-PROCEDURE:ADD-SUPER-PROC(h, SEARCH-TARGET).

RUN s3.p PERSISTENT SET h.
THIS-PROCEDURE:ADD-SUPER-PROC(h).

RUN add-record.
```

If S2 was added with no *proc-search* value (Instance 1, commented out) or with a *proc-search* value of SEARCH-SELF, when RUN SUPER is executed within add-record in S2, the AVM starts searching in S2A, which is next in the search stack of the super procedure, S2.

If S2 was added with a *proc-search* value of SEARCH-TARGET (Instance 2, as executed), when RUN SUPER is executed within add-record in S2, the AVM starts searching in S1, which is next in the search stack of the local procedure, `main.p`.

Note: The search commences with the super procedure following *super-proc-hdl* in the local procedure's search stack.

ADM-DATA attribute

An arbitrary string value associated with a persistent procedure.

Note: The ADM-DATA attribute is for use by the OpenEdge ADM only.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer object handle, Buffer-field object handle, Call object handle, Data-relation object handle, Data-source object handle, Procedure object handle, ProDataSet object handle, Query object handle, SAX-attributes object handle, SAX-reader object handle, SOAP-header object handle, SOAP-header-entryref object handle, SOURCE-PROCEDURE system handle, Stream object handle, TARGET-PROCEDURE system handle, Temp-table object handle, THIS-PROCEDURE system handle

AFTER-BUFFER attribute

Returns the handle to the default buffer of the after-image table that corresponds to the buffer of the before-image table currently associated with this buffer handle.

Data type: HANDLE

Access: Read-only

Applies to: Buffer object handle

See also

BEFORE-BUFFER attribute

AFTER-ROWID attribute

Returns the ROWID of the row in the after-image table that is the current version of the row in the before-image table currently associated with this buffer handle. This row can be a new or modified row.

Data type: ROWID

Access: Read-only

Applies to: Buffer object handle

This attribute is set to the Unknown value (?) for rows that have been deleted.

See also

BEFORE-ROWID attribute

AFTER-TABLE attribute

Returns the handle of the after-image table that corresponds to the before-image table currently associated with this temp-table handle.

Data type: HANDLE

Access: Read-only

Applies to: Temp-table object handle

See also

BEFORE-TABLE attribute

ALLOW-COLUMN-SEARCHING attribute (Windows only)

Setting this attribute to TRUE allows column searching for browses.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

The default is FALSE for read-only static browses. The default is TRUE for dynamic browses and static updateable browses.

If ALLOW-COLUMN-SEARCHING is set to TRUE, the START-SEARCH and END-SEARCH events will be triggered when a search is initiated and completed.

ALLOW-PREV-DESERIALIZATION attribute

Indicates whether to allow an application to deserialize an object using the serialization from the previous OpenEdge major release version.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SECURITY-POLICY system handle

The default value is `NO`. This attribute is provided as a migration path for upgrading applications to the current release of OpenEdge. By default an object serialized in a pre-12.0 release cannot be deserialized in 12.0 and later. For backward compatibility, setting the `ALLOW-PREV-DESERIALIZATION` attribute to `YES`/`TRUE` allows an application to receive an object using the serialization from the previous OpenEdge major release version. Note that passing a serialized object from the current OpenEdge release to an older release is not supported.

For more information, see "R-code updates" in *What's New in OpenEdge 12*.

Note: This attribute affects object serialization when passing objects as parameters over the application server boundary and also via the Progress.IO.BinarySerializer class.

ALWAYS-ON-TOP attribute
(Windows only)

Indicates whether the window should remain on top of all windows, even windows belonging to other applications.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WINDOW widget

This attribute differs from the TOP-ONLY attribute, which indicates that the window should remain on top of all windows of the ABL session. Windows that have the ALWAYS-ON-TOP attribute set are always above TOP-ONLY windows in the z-order; they are also above dialog boxes in some cases.

A window cannot have both the TOP-ONLY and ALWAYS-ON-TOP attributes set. Setting the ALWAYS-ON-TOP attribute to TRUE will set the TOP-ONLY attribute to FALSE. The default value of the ALWAYS-ON-TOP attribute is FALSE.

.NET forms also have a TopMost property that also indicates whether the form should remain on top of all forms of all running applications. Windows allows you to set both an ABL window as an ALWAYS-ON-TOP window and a .NET form as a TopMost form. The window and form will each stay on top of all other windows on the desktop, because Windows maintains two categories of windows—top-most windows and non-topmost windows. All topmost windows appear in front of all non-topmost windows, and you can shuffle the Z-order of the two groups of windows relative to their siblings.

See also

TOP-ONLY attribute

AMBIGUOUS attribute

Indicates whether more than one record matched the FIND predicate.

Data type: LOGICAL

Access: Read-only

Applies to: Buffer object handle

If AMBIGUOUS is TRUE, the most recent unique find on the buffer failed because more than one record matched the FIND predicate. Otherwise, AMBIGUOUS is FALSE.

APPEND-CHILD( ) method

Appends a node as the last child node of this XML document or element node. Connects a node into the document structure after the node has been created with the CREATE-NODE( ) or CREATE-NODE-NAMESPACE( ) method, cloned with the CLONE-NODE( ) method, or disconnected with the REMOVE-NODE( ) method. This has no effect on the node reference.

If the node is already in the tree, it is disconnected from its present location and then connected at the specified location.

Return type: LOGICAL

Applies to: X-document object handle, X-noderef object handle

Syntax

```
APPEND-CHILD ( x-node-handle )
```

x-node-handle

The handle that represents the node to append to this XML document or element node. *x-node-handle* must refer to a node in this document. You cannot use APPEND-CHILD( ) to move a node from one document to another.

The following code fragment demonstrates creating a node in a tree with the name and value of a field:

```
. . .
hDoc:CREATE-NODE(hNoderef, bufField:NAME, "ELEMENT").
hNoderefParent:APPEND-CHILD(hNoderef).
hDoc:CREATE-NODE(hText, "", "TEXT").
hText:NODE-VALUE = STRING(bufField:BUFFER-VALUE).
hNoderef:APPEND-CHILD(hText).
. . .
```

APPL-ALERT-BOXES attribute

Directs application messages to alert boxes or the default message area.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

If the APPL-ALERT-BOXES attribute is TRUE, an application message produced by the MESSAGE statement is displayed in alert boxes rather than in the message area. The default value is FALSE.

APPL-CONTEXT-ID attribute

Returns the universally unique identifier (UUID) for the application context in effect for the current session, as a Base64 character string. The UUID is 22 characters in length (the two trailing Base64 pad characters are removed).

Data type: CHARACTER

Access: Read-only

Applies to: AUDIT-CONTROL system handle

The value of this attribute is set by the SET-APPL-CONTEXT( ) method, and cleared by the CLEAR-APPL-CONTEXT( ) method. This value is recorded in all audit event records generated for this application context until you either clear the current application context or set a different application context.

If no application context is in effect, this method returns the Unknown value (?).

See also

CLEAR-APPL-CONTEXT( ) method, SET-APPL-CONTEXT( ) method

APPLY-CALLBACK( ) method

Applies a callback procedure, which lets you execute a defined event without duplicating the event procedure definition.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle, Query object handle

Syntax

```
APPLY-CALLBACK ( event-name )
```

event-name

The name of a defined event.

Use the SET-CALLBACK-PROCEDURE( ) method to associate an internal procedure with a callback for an object.

For more information on events, see the Handle-based Object Events Reference.

See also

GET-CALLBACK-PROC-CONTEXT( ) method, GET-CALLBACK-PROC-NAME( ) method, SET-CALLBACK-PROCEDURE( ) method

---

APPSERVER-INFO attribute

Connection parameter for the AppServer CONNECT( ) method.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: CODEBASE-LOCATOR system handle

Valid only if LOCATOR-TYPE is "AppServer".

APPSERVER-PASSWORD attribute

Password parameter for the AppServer CONNECT( ) method.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: CODEBASE-LOCATOR system handle

Valid only if LOCATOR-TYPE is "AppServer".

APPSERVER-USERID attribute

Userid parameter for the AppServer CONNECT( ) method.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: CODEBASE-LOCATOR system handle

Valid only if LOCATOR-TYPE is "AppServer".

ASYNCHRONOUS attribute

Makes a dynamic invoke asynchronously. That is, the INVOKE( ) method with ASYNCHRONOUS set to TRUE does dynamically what the RUN statement with the ASYNCHRONOUS option does statically.

Note: If ASYNCHRONOUS is TRUE, the logic being invoked must reside on an application server.

Data type: LOGICAL

Access: Readable/Writable

Applies to: Call object handle

Syntax

```
ASYNCHRONOUS [ = logical-expression ]
```

logical-expression

A LOGICAL expression which, if TRUE, makes the dynamic invoke asynchronous. The default is FALSE.

If ASYNCHRONOUS is TRUE, when the dynamic invoke returns successfully, it sets the ASYNC-REQUEST-HANDLE attribute to indicate an asynchronous-request object that provides information on this particular asynchronous request.

ASYNC-REQUEST-COUNT attribute

The number of active asynchronous requests for the specified procedure or application server.

Data type: INTEGER

Access: Read-only

Applies to: Procedure object handle, Server object handle

For a procedure handle, this attribute is only meaningful if PROXY and PERSISTENT are set to TRUE. In all other cases, this attribute returns zero (0).

The AVM sets this attribute to one (1) on the following handles:

- A proxy procedure handle created for an initial asynchronous request
- The server handle of the application server where the asynchronous remote procedure is instantiated using the PERSISTENT option of the RUN statement.

The AVM increments this attribute:

- On any proxy procedure handle where an internal procedure defined in the specified remote persistent procedure context is executed asynchronously
- On any server handle where an internal or external procedure is executed asynchronously on the specified application server

The AVM decrements this attribute as part of processing the PROCEDURE-COMPLETE event for one of the associated asynchronous requests. The attribute is decremented before any associated event procedure or method is executed in the context of a PROCESS EVENTS, WAIT-FOR, or other input-blocking statement.

ASYNC-REQUEST-HANDLE attribute

A handle to an asynchronous-request object providing information on an asynchronous invoke.

Note: Applies only if ASYNCHRONOUS is TRUE.

Data type: HANDLE

Access: Read Only

Applies to: Call object handle

The default is the Unknown value (?).

ATTACH-DATA-SOURCE( ) method

Attaches a data-source object to a temp-table buffer in a ProDataSet object.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
ATTACH-DATA-SOURCE ( datasource-hdl
  [ [ [ , pairs-list ] , except-fields ] , include-fields ] )
```

datasource-hdl

A handle to the data-source object.

pairs-list

An optional character expression that evaluates to a comma-delimited list of field name pairs that specify a mapping between different field names in the data-source object buffer and the ProDataSet temp-table buffer using the following syntax:

```
"table1-field1,table2-field1 [,table1-fieldn,table2-fieldn ]..."
```

The order within each field name pair does not matter; each pair must contain one field name from the data-source object buffer and one field name from the ProDataSet object buffer. If you have a field in both the data-source object buffer and the ProDataSet object buffer with the same, and you do not want to map them to each other, you must qualify each field name with its table name.

You can use the ATTACHED-PAIRLIST attribute to retrieve this list of field name pairs.

except-fields

An optional character expression that evaluates to a comma-separated list of fields in the ProDataSet object buffer that will not be populated with data from the data source (that is, fields to exclude). Use this option when it is easier to specify fields to exclude rather than include. You can specify *except-fields* or *include-fields*, but not both.

include-fields

An optional character expression that evaluates to a comma-separated list of fields to include in the ProDataSet object buffer, as an alternative to specifying fields to exclude in *except-fields*. Use this option when it is easier to specify fields to include rather than exclude. You can specify *include-fields* or *except-fields*, but not both. If you specify *include-fields*, you must set *except-fields* to the Unknown value (?).

See also

DETACH-DATA-SOURCE( ) method

ATTACHED-PAIRLIST attribute

Returns a comma-separated list of field name pairs for fields in a ProDataSet temp-table buffer that are mapped to corresponding fields in an attached data-source object. This list includes only the field name pairs you specified with the most recently attached data-source object (in the order you specified them).

Data type: CHARACTER

Access: Read-only

Applies to: Buffer object handle

This list is formatted as a comma-delimited list of field name pairs using the following syntax:

Syntax

```
"table1-field1,table2-field1[,table1-fieldn,table2-fieldn ]..."
```

If the buffer is not part of a ProDataSet object, or the buffer does not have an attached data-source object, or you did not specify a field name pair list when you attached the data-source object, this attribute returns the Unknown value (?).

Use the DATA-SOURCE-COMPLETE-MAP attribute to retrieve a list of field name pairs for **all** fields in a ProDataSet temp-table buffer that are mapped to corresponding fields in an attached data-source object.

See also

ATTACH-DATA-SOURCE( ) method

ATTRIBUTE-NAMES attribute

Returns a comma-separated list of an element's attribute names. The attribute names are contained in the XML document. If the element does not have any attributes, the empty string ("") is returned.

Data type: CHARACTER

Access: Read-only

Applies to: X-noderef object handle

If hNoderef is an element node with various attributes, and anames and bname are character program variables, the following example demonstrates listing all the attributes of the XML node:

```
anames = hNoderef:ATTRIBUTE-NAMES.
REPEAT jx = 1 TO NUM-ENTRIES(anames):
  bname = ENTRY(jx, anames).
  MESSAGE "attribute-name is" bname "value is"
    hNoderef:GET-ATTRIBUTE(bname).
END.
```

ATTR-SPACE attribute

This attribute is deprecated.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FILL-IN widget, TEXT widget

AUDIT-EVENT-CONTEXT attribute

The audit event context for a client-principal object. The AVM stores this application-defined audit context in the `_Event-detail` field in the audit record created for an audit event generated during user authentication with the SEAL( ) method, SECURITY-POLICY:SET-CLIENT( ) method, SET-DB-CLIENT function, or AUTHENTICATION-FAILED( ) method, and with the LOGOUT( ) method. If not specified, the `_Event-detail` field in the audit record is left blank.

You can also use this value as an alternate index for querying the audit event record.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

If not already assigned, during any OpenEdge-performed user authentication operation, OpenEdge assigns the value of this attribute (if any) from the domain configuration used to authenticate the asserted identity before sealing the client-principal object.

Once the client-principal object is sealed, this attribute is read-only, and attempting to write to it raises a run-time error.

Also, attempting to set this attribute to a value longer than 200 characters raises a run-time error.

See also

AUTHENTICATION-FAILED( ) method, LOGOUT( ) method, SEAL( ) method, SET-CLIENT( ) method, SET-DB-CLIENT function

AUTHENTICATION-FAILED( ) method

Indicates that the identity asserted in the unsealed client-principal object cannot be authenticated. This authentication failure signifies that the application or database user identity in the registered domain is not authentic. Once invalidated, the client-principal object's properties cannot be changed and the object cannot be sealed. This method also sets the LOGIN-STATE attribute on the client-principal object to `"FAILED"`.

Note: An application can use this method to invalidate an unsealed client-principal object for any reason.

Return type: LOGICAL

Applies to: Client-principal object handle

Syntax

```
AUTHENTICATION-FAILED ( [ reason ] )
```

reason

An optional character expression that specifies the reason for the authentication failure, for example, `"Invalid user name or password"`. The AVM sets the STATE-DETAIL attribute to this value.

If successful, this method returns TRUE. Otherwise, it returns FALSE with any messages returned in the ERROR-STATUS system handle.

Calling this method generates a failed login audit event and creates an audit record for the event in all connected audit-enabled databases according to each database's current audit policy settings.

The SET-DB-CLIENT function and SET-CLIENT( ) method implicitly call this method if authentication fails on the identity asserted for an unsealed client-principal object.

If you call this method on a sealed client-principal object, the AVM raises a run-time error.

The following code fragment illustrates how to use the AUTHENTICATION-FAILED( ) method:

```
DEFINE VARIABLE hCP    AS HANDLE  NO-UNDO.
DEFINE VARIABLE val-ok AS LOGICAL NO-UNDO.
. . .
CREATE CLIENT-PRINCIPAL hCp.
. . .
val-ok = hCP:AUTHENTICATION-FAILED("Invalid username or password").
```

See also

LOGIN-STATE attribute, LOGOUT( ) method, STATE-DETAIL attribute

AUTO-COMPLETION attribute
(Windows only; Graphical interfaces only)

Specifies that the combo-box widget automatically complete keyboard input based on a potential match to items in the drop-down list.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (column), COMBO-BOX widget

When the AUTO-COMPLETION attribute is TRUE, the widget's edit control compares the input to the items in the drop-down list. After each incremental character keystroke, the edit control searches through the items in the drop-down list for a potential match. If a potential match is found, the full item is displayed in the edit control. The automatically completed portion of the item is highlighted. You can replace the highlighted portion of the item by typing over it, or delete the highlighted portion of the item using the **DELETE** key or the **BACKSPACE** key. The default value is FALSE.

For browses, this attribute applies only to combo-box browse columns.

AUTO-DELETE attribute

Specifies whether a dynamic buffer and temp-table object associated with a ProDataSet object is automatically deleted when the ProDataSet object is deleted. Dynamic buffer and temp-table objects associated with a ProDataSet object are deleted when the ProDataSet object is deleted, by default.

Set this attribute to FALSE to prevent a dynamic buffer and temp-table from being automatically deleted when the associated ProDataSet object is deleted. The default value is TRUE.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Buffer object handle

Note: If the buffer is not dynamic and is not associated with a ProDataSet, then AUTO-DELETE returns the Unknown value (?).

AUTO-DELETE-XML attribute

Determines whether the X-document object handle is deleted on a new web request. The default is YES.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

AUTO-END-KEY attribute

Directs the AVM to apply the ENDKEY event to the current frame when a user chooses the button.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BUTTON widget

If the AUTO-END-KEY attribute is TRUE, the AVM applies the ENDKEY event to the frame when the button is chosen. The default value is FALSE.

AUTO-ENDKEY is a synonym for the AUTO-END-KEY attribute.

AUTO-GO attribute

Directs the AVM to apply the GO event to the current frame when a user chooses the button.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BUTTON widget

If the AUTO-GO attribute is TRUE, the AVM applies the GO event to the frame when the button is chosen. The default value is FALSE.

AUTO-INDENT attribute

Specifies the text indentation behavior in the editor widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: EDITOR widget

If AUTO-INDENT is TRUE, each new line of text automatically indents to line up with the preceding line.

AUTO-RESIZE attribute
(Graphical interfaces only)

Tells the AVM how to resize a widget when the LABEL, FONT, or FORMAT attribute of the widget changes.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (column), BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

If the AUTO-RESIZE attribute is TRUE, the widget automatically resizes when the LABEL, FONT or FORMAT attributes of the widget change. If AUTO-RESIZE is FALSE, the widget retains its original size.

The default value for this attribute is TRUE for widgets that are not explicitly sized when they are defined, and FALSE for explicitly sized widgets.

When the AUTO-RESIZE attribute is set to TRUE, the AVM resizes button and toggle-box widgets with run-time changes to the LABEL attribute, and combo-box and fill-in field widgets with run-time changes to the FORMAT attribute.

This attribute resizes the following widgets with run-time changes to the FONT attribute:

- Browse columns
- Buttons
- Combo boxes
- Editors
- Fill-ins
- Radio sets
- Selection lists
- Sliders
- Texts
- Toggle boxes

For browse columns, if you set the browse column's AUTO-RESIZE attribute to TRUE, the AVM resizes the browse column when a change occurs in the browse column's font or in the font or text of the browse column's label.

If the font of a browse column grows such that the height needs to be increased, the AVM increases the height of all cells in the browse column, which increases the row height of the browse (because all rows have the same height). This might affect the DOWN, ROW-HEIGHT-CHARS, and ROW-HEIGHT-PIXELS attributes of the browse column.

If the font of a browse column decreases, the AVM does not decrease the height of the rows, because the decrease might clip text in other columns.

Note: If the developer changes the size of the widget at run time by using the HEIGHT-CHARS, HEIGHT-PIXELS, WIDTH-CHARS, or WIDTH-PIXELS attribute, the AVM resets AUTO-RESIZE to FALSE.

AUTO-RETURN attribute

Specifies the behavior that occurs when a user types the last allowable character in the widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (column), FILL-IN widget

The FORMAT attribute controls the number of characters that a user can enter in the widget. By default, if the user attempts to enter more characters than the number allowed in the widget, the AVM beeps and ignores characters. You can use the AUTO-RETURN attribute to alter this behavior only if the DATA-ENTRY-RETURN attribute of the SESSION handle is TRUE.

If DATA-ENTRY-RETURN and AUTO-RETURN are TRUE and a user types the last character in a field, a LEAVE event occurs and input focus moves to the next widget in the tab order. If the widget is the last widget in the tab order, a GO event occurs for the current frame. This behavior is the same as pressing **RETURN** or **TAB** in the field when the DATA-ENTRY-RETURN attribute is TRUE.

For browse columns, if AUTO-RETURN is TRUE, when the user enters the last allowable character in a browse-cell, the AVM behaves as if the user pressed the **RETURN** key.

AUTO-SYNCHRONIZE attribute

Indicates whether the AVM automatically synchronizes a hierarchy of queries on a ProDataSet temp-table buffer.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Buffer object handle

Set to TRUE to synchronize the related buffers automatically. The default is FALSE.

When AUTO-SYNCHRONIZE is TRUE and a new row is placed in the buffer, the synchronize action occurs. The event handler is called when the buffer has a CREATE, DELETE, RELEASE, FIND, FOR-EACH, QUERY GET, or BUFFER-COPY run on it.

If the BUFFER-COPY is part of a FILL operation, a before-image operation (such as SAVE-ROW-CHANGES), or a deep-copy during parameter passing or COPY-TEMP-TABLE, then the synchronize action does not occur.

If you perform a manual FILL operation using BUFFER-COPY, you can prevent the query hierarchy from being synchronized unnecessarily by setting the AUTO-SYNCHRONIZE attribute to FALSE.

Note: If the buffer is not dynamic and is not associated with a ProDataSet, then AUTO-SYNCHRONIZE returns the Unknown value (?).

AUTO-VALIDATE attribute

Specifies when the AVM runs the validation for a browse column.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

If TRUE, the AVM runs the validation for a browse column in the specified browser on LEAVE of the browse cell. If FALSE, the AVM runs the validation only when code for a browse or browse column specifically invokes the VALIDATE( ) method.

AUTO-ZAP attribute

Specifies what happens to the existing contents of the widget when the user types new information into the widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (cell), COMBO-BOX widget, FILL-IN widget

If AUTO-ZAP is TRUE, when the user begins typing in the field, the entire initial value is erased before the user's text appears. If AUTO-ZAP is FALSE, text entered by the user is inserted into existing text at the current cursor position in the field.

You can set AUTO-ZAP only when the fill-in or cell has input focus (its handle is equal to the FOCUS handle). Otherwise, AUTO-ZAP is TRUE when the user tabs or back-tabs into the field, highlighting text in the field. (When the user selects all text in the field, the same effect occurs without setting the AUTO-ZAP attribute.) AUTO-ZAP is FALSE when the user enters the field with the mouse pointer, positioning the text cursor in the field.

For browses, this attribute applies only to fill-in and combo-box browse columns.

AVAILABLE attribute

Indicates whether a buffer contains a record.

Data type: LOGICAL

Access: Read-only

Applies to: Buffer object handle, Buffer-field object handle

For the buffer object handle, the AVAILABLE attribute corresponds to the AVAILABLE function. If the buffer contains a record, AVAILABLE is TRUE. Otherwise, AVAILABLE is FALSE.

Generally, a buffer-field object handle corresponds to a field returned in a query buffer. However, this field can be excluded from the query using a field list. In this case, if you try to read the BUFFER-VALUE attribute on the associated buffer-field object handle, the AVM returns an error indicating that the corresponding field is missing from the query buffer. You can use the AVAILABLE attribute to test whether the corresponding field was included or excluded from the query.

Depending on its return value, the AVAILABLE attribute indicates one of the following conditions when applied to the buffer-field object:

- **TRUE** — The query buffer has a record with a field available that corresponds to this buffer-field object handle.
- **FALSE** — The query buffer has a record with the field missing that corresponds to this buffer-field object handle.

- **Unknown value (?)** — The query buffer associated with this buffer-field object handle has no record.

AVAILABLE-FORMATS attribute

A comma-separated list of names that specify the formats available for the data currently stored in the clipboard.

Data type: CHARACTER

Access: Read-only

Applies to: CLIPBOARD system handle

If there are no formats available, the attribute returns the Unknown value (?). The supported formats include:

- **PRO_TEXT** — Specifies the standard text format on your system (CF_TEXT in Windows)
- **PRO_MULTIPLE** — Specifies that the data in the clipboard contains tab or newline characters, and thus can be read as multiple items

For more information, see the reference entry for the CLIPBOARD system handle.

BACKGROUND attribute

Specifies the handle for the background iteration of the frame or dialog box.

Data type: HANDLE

Access: Read-only

Applies to: DIALOG-BOX widget, FRAME widget

BASE-ADE attribute

Sets the location of the ADE r-code directory. When set, the AVM adds the directory, followed by all the procedure libraries in the directory, to the PROPATH.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SESSION system handle

You can also specify the ADE r-code location using the ADE R-code Location (-baseADE) startup parameter. For more information about the ADE R-code Location (-baseADE) startup parameter, see *Startup Command and Parameter Reference*.

With the BASE-ADE attribute and the ADE R-code Location (-baseADE) startup parameter, you can have multiple versions of ADE r-code on the same machine and easily switch between them.

If -baseADE is not specified at startup, SESSION:BASE-ADE has the Unknown value (?) until it is set.

When SESSION:BASE-ADE is set, the AVM adds the directory, followed by all of the procedure libraries in the directory to PROPATH. If the directory or any of the procedure libraries are already on PROPATH, the AVM does not add them.

When SESSION:BASE-ADE is set, the AVM removes all PROPATH entries representing the current ADE r-code directory and procedure libraries before adding the new PROPATH entries. The AVM adds the new PROPATH entries at the location where it removed the previous entries. The AVM only removes PROPATH entries that it added. For example, if `$DLC/gui/adecomm.pl` is part of the PROPATH, it remains on the PROPATH after BASE-ADE is set to a directory other than `$DLC/gui`.

If BASE-ADE is set to an empty string, the AVM removes whatever it added to PROPATH.

The AVM does not remove the ADE r-code directory or any of the procedure libraries in that directory from PROPATH, even if the PROPATH statement does not contain them. These entries are part of the base PROPATH. If `-baseADE` or SESSION:BASE-ADE is used, the directory and procedure libraries that the AVM adds are part of the base PROPATH and remain part of the PROPATH even if the PROPATH statement does not contain them.

SESSION:BASE-ADE modifies PROPATH. If the old PROPATH contains a procedure library that is not in the new PROPATH, the AVM automatically closes the procedure library as long as there are no procedures from the library running.

In Windows, the r-code directory name cannot contain characters outside of the non-Unicode code page. See *Internationalize ABL Applications* for more information about Unicode and code pages.

BASIC-LOGGING attribute

Turns on `QryInfo` logging for an individual query.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Query object handle

ABL does not provide `QryInfo` logging for individual queries at logging level 2 (Basic), by default. You must use the BASIC-LOGGING attribute to turn on logging for an individual query when the logging level of the `QryInfo` log entry type is set to level 2 (Basic). If the logging level of the `QryInfo` log entry type is level 3 (Verbose) or higher, then ABL ignores any BASIC-LOGGING attribute setting and logs query information for all queries.

To set the logging level of the `QryInfo` log entry type, use the LOG-ENTRY-TYPES attribute or the Log Entry Types (`-logentrytypes`) startup parameter with the logging level option.

To turn on logging for an individual query when the logging level is set to 2 (Basic), you must set this attribute to TRUE before a query starts. For a dynamically opened query, this is before the QUERY-PREPARE( ) method. For a statically opened query, this is before the OPEN QUERY statement. If you set this attribute to TRUE after a query starts, ABL does not provide logging for that query.

To turn off logging for an individual query when the logging level is set to 2 (Basic), you must set this attribute to FALSE. If you set this attribute to FALSE before a query completes, ABL does not write query statistics to the log.

Whenever you turn on or turn off logging for an individual query using this attribute, ABL writes a log entry to the log file indicating the query ID, the query object handle, and the name of the query.

For more information about the Log Entry Types (`-logentrytypes`) startup parameter, see *Startup Command and Parameter Reference*.

For more information about query logging, see *Troubleshoot ABL Applications*.

BATCH-MODE attribute

Indicates whether the current ABL session is running in batch mode or interactive mode.

Data type: LOGICAL

Access: Read-only

Applies to: SESSION system handle

BATCH-MODE is TRUE when the current session is running in batch or background mode. A session runs in batch or background mode when it is:

- A client session started using the Batch (-b) startup parameter
- An application server agent session

BATCH-SIZE attribute

The maximum number of ProDataSet temp-table rows to retrieve in each FILL operation. The default value is zero (which retrieves all rows that satisfy the associated query).

Note: If you specify a batch size for a ProDataSet temp-table that is a child of a relation, the AVM restarts the BATCH-SIZE counter for each parent record (as opposed to once per temp-table).

Data type: INTEGER

Access: Readable/Writeable

Applies to: Buffer object handle

This attribute is not marshalled between the client and the AppServer (unlike the LAST-BATCH attribute).

See also

FILL( ) method, LAST-BATCH attribute

BEFORE-BUFFER attribute

Returns the handle to the default buffer of the before-image table that corresponds to the buffer of the after-image table currently associated with this buffer handle.

Data type: HANDLE

Access: Read-only

Applies to: Buffer object handle

See also

AFTER-BUFFER attribute

BEFORE-ROWID attribute

Returns the ROWID of the row in the before-image table that corresponds to the row in the after-image table currently associated with this buffer handle.

Data type: ROWID

Access: Read-only

Applies to: Buffer object handle

This attribute is set to the Unknown value (?) for row that have not changed.

See also

AFTER-ROWID attribute

BEFORE-TABLE attribute

Returns the handle of the before-image table that corresponds to the after-image table currently associated with this temp-table handle.

Data type: HANDLE

Access: Read-only

Applies to: Temp-table object handle

See also

AFTER-TABLE attribute

BEGIN-EVENT-GROUP( ) method

Indicates (and records) the beginning of a group of related audit events in the current session. Audit event groups are used to group a series of related application and database audit events in one or more connected audit-enabled databases whose current audit policy has this audit event enabled.

This method returns a Base64 character string that specifies the universally unique identifier (UUID) of the primary index for all audit event records generated by this method for this audit event group. This UUID is recorded in all subsequent audit event records until you either end this audit event group or begin a different audit event group. The UUID is 22 characters in length (the two trailing Base64 pad characters are removed).

Return type: CHARACTER

Applies to: AUDIT-CONTROL system handle

Syntax

```
BEGIN-EVENT-GROUP( event-context [ , event-detail
   [ , audit-custom-detail ] ] )
```

event-context

A character expression that specifies the context for the audit event. The value of this expression cannot exceed 200 characters. You can also use this value as an alternate index for querying the audit event record.

If you specify the Unknown value (?), the AVM generates a run-time error.

event-detail

An optional character expression that specifies additional audit detail. The value of this expression cannot exceed 10,000 characters.

audit-custom-detail

An optional character expression that specifies additional user detail. The value of this expression cannot exceed 10,000 characters.

The UUID is saved as the EVENT-GROUP-ID attribute value for each connected audit-enabled database.

There can be only one active event group per session at any one point in time. To set a different event group for the session, you can:

- Call the END-EVENT-GROUP( ) method, to end the current event group, and then call the BEGIN-EVENT-GROUP( ) method to begin the new event group.
- Call the BEGIN-EVENT-GROUP( ) method to begin the new event group. If there is an existing event group in effect, the AVM ends the existing event group before beginning the new event group.

Calling this method generates an audit event, and creates an audit record for the event in all connected audit-enabled databases according to each database's current audit policy settings.

The following code fragment illustrates how to use the BEGIN-EVENT-GROUP( ) method:

```
DEFINE VARIABLE name   AS CHARACTER NO-UNDO.
DEFINE VARIABLE ctx-id AS CHARACTER NO-UNDO.
DEFINE VARIABLE grp-id AS CHARACTER NO-UNDO.
. . .
grp-id = AUDIT-CONTROL:BEGIN-EVENT-GROUP
  ("Payroll app", "tax calculations", name).
ctx-id = AUDIT-CONTROL:SET-APPL-CONTEXT
  ("Payroll app", "federal tax calculation", name).
. . .
AUDIT-CONTROL:LOG-AUDIT-EVENT(34122, "payroll.fed.tax.nh").
. . .
ctx-id = AUDIT-CONTROL:SET-APPL-CONTEXT
  ("Payroll app", "fica calculation", name).
. . .
AUDIT-CONTROL:LOG-AUDIT-EVENT(34123, "payroll.fed.tax.ma").
. . .
AUDIT-CONTROL:CLEAR-APPL-CONTEXT.
AUDIT-CONTROL:END-EVENT-GROUP.
```

See also

END-EVENT-GROUP( ) method, EVENT-GROUP-ID attribute

BGCOLOR attribute
(Graphical interfaces only)

Specifies the color number for the background color of the widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse, column, and cell), COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

For a rectangle, if the FILLED attribute is TRUE, BGCOLOR specifies the color of the region inside the border of the rectangle.

For a browse cell, BGCOLOR specifies the color of a specific cell in the view port. You can set this color only as the cell appears in the view port during a ROW-DISPLAY event.

You cannot set this attribute for a BUTTON widget.

Setting the BGCOLOR attribute for any of the following field-level widgets within a dialog-box or frame overrides any INHERIT-BGCOLOR attribute or option settings: BROWSE, COMBO-BOX (all types), EDITOR, FILL-IN (NATIVE and Enabled), and SELECTION-LIST.

See also

FGCOLOR attribute, Frame phrase, INHERIT-BGCOLOR attribute

BLANK attribute

Suppresses the display of sensitive data in a field.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FILL-IN widget, TEXT widget

If the BLANK attribute is TRUE, any current value or characters typed in the fill-in are not echoed to the screen. The default value for this attribute is FALSE.

BLOCK-ITERATION-DISPLAY attribute

Specifies if the Frame phrase of the frame contains the NO-HIDE option or if the frame has multiple iterations (is a DOWN frame).

Data type: LOGICAL

Access: Read-only

Applies to: FRAME widget

The BLOCK-ITERATION-DISPLAY attribute returns TRUE if the NO-HIDE option is specified in a frame phrase for the frame or the frame has multiple iterations.

BORDER-BOTTOM-CHARS attribute

The thickness, in character units, of the border at the bottom of the frame or dialog box.

Data type: DECIMAL

Access: Read-only

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BORDER-BOTTOM-PIXELS attribute

The thickness, in pixels, of the border at the bottom of the frame or dialog box.

Data type: INTEGER

Access: Read-only

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BORDER-LEFT-CHARS attribute

The thickness, in character units, of the border at the left side of the frame or dialog box.

Data type: DECIMAL

Access: Read-only

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BORDER-LEFT-PIXELS attribute

The thickness, in pixels, of the border at the left side of the frame or dialog box.

Data type: INTEGER

Access: Read-only

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BORDER-RIGHT-CHARS attribute

The thickness, in character units, of the border at the right side of the frame or dialog box.

Data type: DECIMAL

Access: Read-only

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BORDER-RIGHT-PIXELS attribute

The thickness, in pixels, of the border at the right side of the frame or dialog box.

Data type: INTEGER

Access: Read-only

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BORDER-TOP-CHARS attribute

The thickness, in character units, of the border at the top of the frame or dialog box.

Data type: DECIMAL

Access: Read-only

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BORDER-TOP-PIXELS attribute

The thickness, in pixels, of the border at the top of the frame or dialog box.

Data type: INTEGER

Access: Read-only

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BOX attribute

Indicates whether the widget has a graphical border around it.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: EDITOR widget, FRAME widget

If the BOX attribute is FALSE, the widget does not have a border. You can set this attribute only before the widget is realized.

For editors, BOX has no effect on the size of the editor.

BOX-SELECTABLE attribute
(Graphical interfaces only)

Indicates whether box-selection direct manipulation events for the frame or dialog box are enabled or disabled.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget

If the BOX-SELECTABLE attribute is TRUE, then the START-BOX-SELECTION and END-BOX-SELECTION direct manipulation events are enabled for the frame or dialog box. This allows the user to select one or more widgets in the frame or dialog box by stretching a select box around the widgets. The SELECTABLE attribute must be TRUE for at least one widget in the frame or dialog box for this attribute to be effective. Otherwise, the user can stretch a select box, but without any effect on the widgets in the frame or dialog box.

BUFFER-CHARS attribute
(Character interfaces only)

The number of characters a user can enter on each line of the editor.

Data type: INTEGER

Access: Readable/Writeable

Applies to: EDITOR widget

When the last character is typed on a line in the editor, the text input cursor automatically wraps to the next line. This attribute can also set the word wrap margin for the WORD-WRAP attribute. For more information, see the WORD-WRAP attribute reference entry.

BUFFER-COMPARE( ) method

This method does a rough compare of any common fields, determined by name, data type, and extent-matching, between the source buffer and the target buffer. The resulting logical value is either TRUE or FALSE as a whole. A single field that does not compare causes the entire buffer to return FALSE. If there are fields in one buffer that do not exist in the other, they are ignored.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
BUFFER-COMPARE ( source-buffer-handle [ , mode-exp [ , except-list
  [ , pairs-list [ , no-lobs ] ] ] ] )
```

source-buffer-handle

An expression that evaluates to a buffer handle.

mode-exp

If *mode-exp* is given, it must evaluate to either "binary" or "case-sensitive" to provide that type of comparison.BUFFER-COMPARE( ) method supports binary and case-sensitive comparisons between CLOB as well as CHARACTER fields.

except-list

A character expression that evaluates to a comma-separated list of field names to be excluded from the compare.

pairs-list

A character expression that evaluates to a comma-delimited list of field-name pairs to be compared.

You can specify an array element as one or both of the fields. This lets you compare a field or array element in one buffer to a field or array element in the other buffer, when the two fields do not have the same name. The order within each field-name pair does not matter; each pair must contain one field name from the source and one field name from the target.

You can also compare one entire array to another by specifying its name without a subscript.

no-lobs

A logical expression indicating whether to ignore BLOB and CLOB fields in the compare. If TRUE, BLOB and CLOB fields are ignored during the compare. If FALSE, BLOB and CLOB fields are compared along with the other fields. The default value is FALSE (that is, BLOB and CLOB fields are included in the compare).

Note: You cannot use the BUFFER-COMPARE( ) method to compare records that contain CLOB fields, unless one or both of the corresponding fields contain the Unknown value (?); the AVM generates a run-time error. However, you can convert CLOB fields to LONGCHAR values and use the EQ, GE, GT, LE, LT, and NE comparison operators, or the COMPARE function, to compare the LONGCHAR values.

If you want to compare BLOB fields only, you can set this option to FALSE and use the *except-list* option to exclude CLOB fields from the compare.

Caution: When using this option, you get no warning if a LOB field has changed. This can lead to inappropriate results. Therefore, before using this option, you must understand the nature of your data and be sure that logic using this option will not result in inconsistent or out-of-date data in the database.

When comparing buffers in a ProDataSet object, the AVM checks as to whether the BUFFER-COMPARE( ) method satisfies the following two requirements:

- The compare is between a buffer on a data-source object table and the corresponding ProDataSet temp-table buffer. This means the operation can use any buffer for the data source database table, but only the default buffer for the ProDataSet temp-table.
- There are no *except-list* or *pairs-list* arguments for the BUFFER-COMPARE( ) method.

If these two requirements are satisfied, the BUFFER-COMPARE( ) method identifies the fields to compare based on the *pairs-list* argument specified in the ATTACH-DATA-SOURCE( ) method for the data-source object, if any, along with either the *except-list* or *include-list* arguments, if any. Because the ATTACH-DATA-SOURCE( ) method already allows you to define a field mapping between a data-source object buffer and a ProDataSet temp-table buffer, as well as a list of fields to include or exclude from the operation, you do not need to specify these in the BUFFER-COMPARE( ) method.

When comparing records that contain BLOB fields, the AVM performs a binary comparison on the BLOB data associated with the source and target records.

Use the *no-lobs* option with the BUFFER-COMPARE( ) method to ignore large object data when comparing records that contain BLOB or CLOB fields. You can also use the *except-list* option to exclude BLOB and CLOB fields from the compare.

The following example fragment performs a binary compare of all fields in the buffers, except for the two specified fields:

```
BUFFER-COMPARE(bh2, "binary", "cust-sales-rep,SalesRep").
```

BUFFER-COPY( ) method

This method copies any common fields, determined by name, data type, and extent-matching, from the source buffer to the receiving buffer. If there are fields in one buffer that do not exist in the other, they are ignored. This method is used to accommodate temp-tables of joins.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
BUFFER-COPY ( source-buffer-handle
  [ , except-list [ , pairs-list [ , no-lobs ] ] ] )
```

source-buffer-handle

An expression that evaluates to the source buffer handle.

except-list

A character expression that evaluates to a comma-separated list of field names to be excluded from the copy.

pairs-list

A character expression that evaluates to a comma-separated list of field-name pairs to be copied.

You can also specify an array element as one or both of the fields. This lets you copy a field or array element from one buffer to a field or array element in the other buffer, when the two fields do not have the same name. The order within each field-name pair does not matter; each pair must contain one field name from the source and one field name from the target.

You can also copy one entire array to another by specifying its name without a subscript.

no-lobs

A logical expression indicating whether to ignore BLOB and CLOB fields in the copy. If TRUE, BLOB and CLOB fields are ignored during the copy. If FALSE, BLOB and CLOB fields are copied along with the other fields. The default value is FALSE (that is, BLOB and CLOB fields are included in the copy).

Caution: Using this option can create the potential for errors in your data and lead to inappropriate results. Therefore, before using this option, you must understand the nature of your data and be sure that logic using this option will not result in inconsistent or out-of-date data in the database.

When copying buffers in a ProDataSet object, the AVM checks as to whether the BUFFER-COPY( ) method satisfies the following two requirements:

- The copy is between a buffer on a data-source object table and the corresponding ProDataSet temp-table buffer. This means the operation can use any buffer for the data source database table, but only the default buffer for the ProDataSet temp-table.
- There are no *except-list* or *pairs-list* arguments for the BUFFER-COPY( ) method.

If these two requirements are satisfied, the BUFFER-COPY( ) method identifies the fields to copy based on the *pairs-list* argument specified in the ATTACH-DATA-SOURCE( ) method for the data-source object, if any, along with either the *except-list* or *include-list* arguments, if any. Because the ATTACH-DATA-SOURCE( ) method already allows you to define a field mapping between a data-source object buffer and a ProDataSet temp-table buffer, as well as a list of fields to include or exclude from the operation, you do not need to specify these in the BUFFER-COPY( ) method.

When copying records that contain a BLOB or CLOB field, the AVM copies the object data associated with the source record to the target record. If the BLOB or CLOB field in the source record contains the Unknown value (?), the AVM stores the Unknown value (?) in the BLOB or CLOB field of the target record. If the target record already has object data associated with it, the AVM deletes that object data before copying the new object data.

Use the *no-lobs* option with the BUFFER-COPY( ) method to ignore large object data when copying records that contain BLOB or CLOB fields. More specifically:

- When you copy a source record to a new target record, the AVM sets the value of the BLOB or CLOB field in the target record to the Unknown value (?).
- When you copy a source record to an existing target record, the AVM does not change the value of the BLOB or CLOB field in the existing target record.

You can also use the *except-list* option to exclude BLOB and CLOB fields from the copy.

The following example fragment copies the Customer table to the buffer, *bh*, except that `Customer.SalesRep` is copied to a field called *cust-sales-rep* in the buffer:

```
bh:BUFFER-COPY(BUFFER Customer:HANDLE, ?, "cust-sales-rep,SalesRep").
```

BUFFER-CREATE( ) method

Creates a record, sets fields to their default values, and moves a copy of the record into the buffer.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
BUFFER-CREATE ( [ tenant-expression ] )
```

tenant-expression

This parameter is useful only for a multi-tenant database, and primarily one with a connection identity that has super tenant access. If the user has a super-tenant connection identity and you **do not** specify this parameter, the record you create is owned by the effective tenant. If you **do** specify this parameter, you create a record owned by the regular tenant identified by *tenant-expression*.

If the user has a regular-tenant connection identity, and you specify this parameter, *tenant-expression* must match the tenancy of the connection identity. Otherwise, the statement raises ERROR.

If *tenant-expression* evaluates to an integer, the value must be a valid tenant ID for a regular tenant or zero (0) for the default tenant. If *tenant-expression* evaluates to a character string, the value must be a valid tenant name for a regular or "Default" for the default tenant. Otherwise, the statement raises ERROR.

If Buffer object handle references a buffer in a table that is not multi-tenant enabled, the method raises ERROR.

Notes

- The BUFFER-CREATE( ) method corresponds to the CREATE statement.
- When working with a partitioned table, creating a record without initially setting the partition fields to valid values may cause an error. See CREATE statement for more details.
- When you pass *tenant-expression*, the AVM looks up the tenant identity in the database with a share lock. The AVM waits 60 seconds to get the share lock and raises ERROR if it fails to obtain the share lock in that amount of time. The AVM releases the share lock immediately after successfully fetching the row. This share lock is released even if the statement is called while in the scope of a transaction.

BUFFER-DELETE( ) method

Deletes a record from the record buffer and from the database.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
BUFFER-DELETE ( )
```

The BUFFER-DELETE method corresponds to the DELETE statement.

If the table has delete validation—that is, if the table specifies an expression that must be true before the record is deleted—the record is not deleted, because the validation expression, normally applied at compile time, cannot be applied fully at run time.

BUFFER-FIELD attribute

The handle of the browse column's buffer-field.

Data type: HANDLE

Access: Read-only

Applies to: BROWSE widget (column)

If the browse column does not have a corresponding buffer field, the Unknown value (?) will be returned.

BUFFER-FIELD( ) method

Returns a handle to a particular field in the buffer.

Return type: HANDLE

Applies to: Buffer object handle

Syntax

```
BUFFER-FIELD ( field-number | field-name )
```

field-number

An integer expression representing the sequence number of the field in the buffer.

field-name

A CHARACTER string expression representing the name of the field in the buffer.

BUFFER-GROUP-ID attribute

The group ID (as an integer) of the tenant group to which the current record in the buffer specified by the buffer handle belongs. If the buffer does not contain a record from a tenant group, the function returns the Unknown value (?).

Data type: INTEGER

Access: Read-only

Applies to: Buffer object handle

Note: This attribute can be used in a WHERE or TENANT-WHERE option as long as the specified buffer is not the same as the buffer of the query or FOR EACH statement.

See also

BUFFER-GROUP-ID function, BUFFER-GROUP-NAME attribute

BUFFER-GROUP-NAME attribute

The name (as a character string) of the tenant group to which the current record in the buffer specified by the buffer handle belongs. If the buffer does not contain a record from a tenant group, the function returns the Unknown value (?).

Data type: CHARACTER

Access: Read-only

Applies to: Buffer object handle

Note: This attribute can be used in a WHERE or TENANT-WHERE option as long as the specified buffer is not the same as the buffer of the query or FOR EACH statement.

See also

BUFFER-GROUP-ID attribute, BUFFER-GROUP-NAME function

BUFFER-HANDLE attribute

The handle of the buffer object to which the buffer-field belongs.

Data type: HANDLE

Access: Read-only

Applies to: Buffer-field object handle

BUFFER-LINES attribute
(Character interfaces only)

The number of lines a user can enter into the editor.

Data type: INTEGER

Access: Readable/Writeable

Applies to: EDITOR widget

You can set this attribute only before the editor widget is realized. The value must be an integer expression that is equal to or greater than the value specified by the HEIGHT-CHARS or INNER-LINES attributes. If equal, vertical scrolling is disabled.

By default, the AVM does not limit the number of enterable lines (although system limits may apply).

BUFFER-NAME attribute

The name of the buffer object to which the buffer-field object belongs.

Data type: CHARACTER

Access: Read-only

Applies to: Buffer-field object handle

BUFFER-PARTITION-ID attribute

The partition ID of the partition of the buffer.

Data type: INTEGER

Access: Read-only

Applies to: Buffer object handle

If the buffer is not populated with a record, this attribute returns the Unknown value (?). If there is a record in the buffer but it is not for a partitioned table, the attribute returns zero (0).

Notes

- If you use this attribute in a WHERE clause, it will be evaluated before the FOR EACH starts, which will result in no records being found unless the specified buffer is not the same as the buffer of the query or FOR EACH statement.
- The partition ID is only unique within a given table.

See also

BUFFER-PARTITION-ID function, IS-PARTITIONED attribute

BUFFER-RELEASE( ) method

Releases a record from a buffer object. The BUFFER-RELEASE method corresponds to the RELEASE statement.

Note: To delete the buffer object, use the DELETE OBJECT statement.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
BUFFER-RELEASE ( )
```

If a record has been modified, this method causes a WRITE event and executes all related WRITE triggers.

If successful, this method returns TRUE. Otherwise, it returns FALSE. If the validation fails on a newly-created record, this method returns FALSE and raises the ERROR condition.

BUFFER-TENANT-ID attribute

The tenant ID (as an integer) of the tenant that owns the current record in the buffer specified by a buffer handle.

Data type: INTEGER

Access: Read-only

Applies to: Buffer object handle

If the buffer is not populated with a record, this attribute returns the Unknown value (?). If the buffer is for a shared table, a temp-table, or for the default tenant of a multi-tenant table, this attribute returns the value zero (0).

If the buffer contains a record for a tenant group and the user is a super tenant, this attribute returns the tenant ID of one of the members of the group. The tenant ID returned is indeterminate, but is always the tenant ID of a tenant in the group.

For a regular tenant, the record in the buffer always belongs to that tenant. So, this attribute always returns the user's own tenant ID. This is true even if the record belongs to a tenant group.

Notes

- This attribute can be used in a WHERE or TENANT-WHERE option as long as the specified buffer is not the same as the buffer of the query or FOR EACH statement.
- This attribute can be used by database triggers on the buffers passed to the trigger to get the tenant ID of these buffers. This allows the application provider to code tenant-specific database triggers.

See also

BUFFER-TENANT-ID function, BUFFER-TENANT-NAME attribute

BUFFER-TENANT-NAME attribute

The name of the tenant (as a character string) that owns the current record in the buffer specified by a buffer handle.

Data type: CHARACTER

Access: Read-only

Applies to: Buffer object handle

If the buffer is not populated with a record, this attribute returns the Unknown value (?). If the buffer is for a shared table or a temp-table, this attribute returns the empty string ("").

If the buffer contains a record that belongs to a tenant group and the user is a super tenant, this attribute returns the tenant name of one of the members of the group. The tenant name returned is indeterminate, but is always the name of a tenant in the group.

For a regular tenant user, the record in the buffer always belongs to that tenant. So, the attribute always returns the user's own tenant name. This is true even if the record belongs to a tenant group.

Note: This attribute can be used in a WHERE or TENANT-WHERE option as long as the specified buffer is not the same as the buffer of the query or FOR EACH statement.

See also

BUFFER-TENANT-NAME function

BUFFER-VALIDATE( ) method

Verifies that a record in a buffer object complies with mandatory field and unique index definitions. The BUFFER-VALIDATE( ) method corresponds to the VALIDATE statement.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
BUFFER-VALIDATE ( )
```

If a field or table has been modified, this method causes a WRITE event and executes all related WRITE triggers.

If successful, this method returns TRUE. Otherwise, it returns FALSE. If the validation fails on a newly-created record, this method returns FALSE and raises the ERROR condition.

The record is not released and the lock status remains the same as before the BUFFER-VALIDATE( ).

BUFFER-VALUE( ) method

The current value of a buffer-field object. If you modify the BUFFER-VALUE( ) method, the AVM sets the buffer-field object to the new value.

Data type: The data type of the corresponding buffer-field

Access: Readable/Writeable

Applies to: Buffer-field object handle

Syntax

```
BUFFER-VALUE ( [ i ] )
```

i

An integer expression representing a subscript, for fields that have extents.

The syntax for retrieving the value of a buffer-field object using a dynamic reference to a table field in a dynamic ProDataSet, temp-table, query, or buffer object can be awkward, especially when you know the table and field names at compile time. ABL provides a simpler way to express the same syntax, only in a short-hand form. For example, following is the typical syntax for referring to the `CustNum` buffer field in the `Customer` table through a ProDataSet handle:

```
hDSet:GET-BUFFER-HANDLE("ttcust"):BUFFER-FIELD("CustNum"):BUFFER-VALUE( )
```

Following is the short-hand form of the same syntax:

```
hDSet::ttcust::CustNum
```

Note: Progress Software Corporation recommends that you do not set the call object's CALL-TYPE attribute to SET-ATTR-CALL-TYPE to set a BUFFER-FIELD object's BUFFER-VALUE ( ) method return value, since there in no way to run triggers for the target field.

See also

LITERAL-QUESTION attribute

BYTES-READ attribute

Returns the number of bytes read from the socket via the last READ( ) method. If the last READ( ) method failed, this attribute will return 0.

Data type: INTEGER

Access: Read-only

Applies to: Socket object handle

BYTES-WRITTEN attribute

Returns the number of bytes written to the socket via the last WRITE( ) method. If the last WRITE( ) method failed, this attribute will return 0.

Data type: INTEGER

Access: Read-only

Applies to: Socket object handle

CACHE attribute

Specifies how many records a NO-LOCK query should hold in memory.

Data type: INTEGER

Access: Readable/Writeable

Applies to: Query object handle

Note

The CACHE attribute corresponds to the CACHE option of the DEFINE QUERY statement.

CALL-NAME attribute

The name of one of the following:

- An external procedure, internal procedure, or user-defined function you want to invoke dynamically
- A Windows DLL routine or Unix shared library routine you want to invoke dynamically
- An attribute you want to get or set dynamically
- A method you want to run dynamically

Data type: CHARACTER

Access: Readable/Writable

Applies to: Call object handle

Syntax

```
CALL-NAME [ = char-expression ]
```

char-expression

A CHARACTER expression indicating the name of the procedure, user-defined function, attribute, method, or shared library routine. The default is the Unknown value (?). In Windows, the expression cannot contain characters outside of the non-Unicode code page. See *Internationalize ABL Applications* for more information about Unicode and code pages.

If *char-expression* is a procedure, the rules for finding the procedure are the same as those of the RUN statement, including the treatment of SUPER. However, the actual instance of the file, procedure name, or attribute is not determined until the dynamic invoke is executed.

If *char-expression* is a user-defined function, an attribute, or a method, the rules for finding the function or attribute are the same as those for finding a function or attribute invoked statically.

If *char-expression* is a shared library routine, at least the following attributes must be set:

- CALL-NAME or ORDINAL
- CALL-TYPE

- LIBRARY
- RETURN-VALUE-DLL-TYPE

If *char-expression* is a shared library routine and LIBRARY specifies a full path name, the AVM searches only that path for the shared library. Otherwise, the AVM first determines if the shared library is loaded in memory. If it is not loaded in memory, the AVM looks for the shared library in the current working directory. If the shared library cannot be found at this point:

- On Windows the AVM uses the standard Windows search algorithm to find the DLL, the same algorithm used to find Windows executable (`.exe`) files
- On UNIX the AVM continues searching for the shared library using the ABL PROPATH

Using the call object handle, there are two alternate methods for invoking a DLL routine—either by specifying the routine's name with the CALL-NAME attribute or by specifying the routine's position in the DLL with the ORDINAL attribute. You can specify an ORDINAL value or a CALL-NAME value, but not both. Specifying an ORDINAL value and setting CALL-NAME at the same time causes the AVM to raise error at run time.

See also

ORDINAL attribute

CALL-TYPE attribute

The type of call, which must be the dynamic version of one of the following:

- Invoking an external or internal procedure
- Invoking a user-defined function
- Invoking a Windows DLL routine or a Unix shared library routine
- Getting an attribute
- Setting an attribute
- Invoking a method

Data type: INTEGER

Access: Readable/Writable

Applies to: Call object handle

You can use a keyword constant or an integer. The following table lists the keyword constants and integer values you can use.

Table 66: Keyword constants for the CALL-TYPE attribute

Keyword constant	Integer value
PROCEDURE-CALL-TYPE	1
FUNCTION-CALL-TYPE	2
GET-ATTR-CALL-TYPE	3

Keyword constant	Integer value
SET-ATTR-CALL-TYPE	4
DLL-CALL-TYPE	5

The default is PROCEDURE-CALL-TYPE.

To get the value of an attribute or to invoke a method, set CALL-TYPE to GET-ATTR-CALL-TYPE.

In this example, the SESSION:NUMERIC-FORMAT attribute is assigned, indicated by the CALL-TYPE value of SET-ATTR-CALL-TYPE:

```
/* Set SESSION:NUMERIC-FORMAT to "european" */

ASSIGN
  hCall:IN-HANDLE      = "session"
  hCall:CALL-TYPE      = SET-ATTR-CALL-TYPE
  hCall:CALL-NAME      = "numeric-format"
  hCall:NUM-PARAMETERS = 1.

hCall:SET-PARAMETER(1, "CHARACTER", "INPUT", "european").
hCall:INVOKE.
```

Note: PSC recommends that you do not set the call object's CALL-TYPE attribute to SET-ATTR-CALL-TYPE to set a BUFFER-FIELD object's BUFFER-VALUE attribute, since there is no way to run triggers for the target field.

CANCEL-BREAK( ) method

Cancels a breakpoint from a debugging session.

Return type: LOGICAL

Applies to: DEBUGGER system handle

Syntax

```
CANCEL-BREAK ( [ procedure [ , line-number ] ] )
```

procedure

A character expression that specifies the name of the procedure for which you want to cancel a breakpoint. The specified procedure does not have to exist at the time the breakpoint is cancelled. If you do not specify *procedure*, the method cancels any breakpoint set on the line immediately following the current line. (This is different from the SET-BREAK( ) method, which sets a breakpoint on the next executable line.)

line-number

An integer expression that specifies the line number in *procedure* (based at line 1 of the debug listing) where you want to cancel the breakpoint. A positive integer greater than or equal to 1 represents a line number in the specified *procedure* file. Zero (0) or a negative integer value represents the first executable line of the main procedure block in the specified *procedure* file. If you do not specify *line-number*, the method cancels the breakpoint at the first executable line of *procedure* file.

If you invoke DEBUGGER:CANCEL-BREAK (*procedure* , *line-number*) on the same line that is specified by *procedure* and *line-number*, the existing breakpoint on the specified line occurs the first time it is executed. The breakpoint is cancelled only on the second and succeeding executions of the line.

To use this method, you must have the Application Debugger installed in your OpenEdge environment.

CANCEL-BUTTON attribute

A button widget in the frame or dialog box to receive the CHOOSE event when a user cancels the current frame or dialog box by pressing the **ESC** key.

Data type: HANDLE

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget

Any other action normally associated with the **ESC** key is not performed. The **ESC** key is any key associated with the **ESC** key label.

The CANCEL-BUTTON attribute for frames is not supported in character mode.

If the user presses ESCAPE in a frame that has no such cancel button and the frame is part of a frame family, the AVM applies the CHOOSE event to the first cancel button it finds within the frame family in random order.

CANCEL-REQUESTS( ) method

For a state-reset, state-aware, or stateless application server, this method raises a STOP condition in the context of the currently running asynchronous request and purges the send queue of any asynchronous requests that have not been executed on the specified application server.

For a state-free application server, this method raises a STOP condition for all currently running asynchronous requests, and purges the send queue of any asynchronous requests that have not been executed on the specified application server.

For Web services, this method terminates the connection to all currently running asynchronous requests and purges the send queue of all asynchronous requests that have not been executed on the specified Web service.

Return type: LOGICAL

Applies to: Server object handle

Syntax

```
CANCEL-REQUESTS( )
```

After executing this method, at the next input-blocking state (or on executing the PROCESS EVENTS statement) event procedures or methods execute for the following asynchronous requests:

- All requests that were complete when this method executed but whose event procedures or methods have not been run.
- All currently running requests that were stopped in response to this method.
- All requests that were purged from the send queue, and never run.

This method returns FALSE when the server handle is not in the connected state. (See the CONNECTED( ) method). Otherwise, this method returns TRUE.

See also

CANCEL-REQUESTS-AFTER( ) method

CANCEL-REQUESTS-AFTER( ) method

Calls the CANCEL-REQUESTS-AFTER( ) method on the server object after the specified number of seconds have elapsed. When CANCEL-REQUESTS-AFTER( ) is called, all requests currently running or queued to run on the server object are cancelled, regardless of when they were started.

If the method is called on a server object that already has a timer running, the timer is restarted with the new time interval, measured from that moment.

If the method is called with a parameter whose value is less than or equal to zero, then the timer on that server object is stopped.

Return type: LOGICAL

Applies to: Server object handle

Syntax

```
CANCEL-REQUESTS-AFTER ( seconds )
```

seconds

This required integer parameter specifies the number of seconds before the CANCEL-REQUESTS-AFTER( ) method is called on the server object.

The method returns TRUE unless an error occurs while setting the timer. In that case, the method returns FALSE, and the timeout is not set.

Example

The following simple example uses the CANCEL-REQUESTS-AFTER( ) method:

```
DEFINE VARIABLE hSrv as HANDLE.
DEFINE VARIABLE bool as BOOLEAN.

CREATE SERVER hSrv.
hSrv:CONNECT("-URL AppServerDC://hostName/svc").

bool = hSrv:CANCEL-REQUESTS-AFTER(30).
RUN foo.p  ON SERVER hSrv ASYNCHRONOUS ("Hello World").

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

Calling this method on a server object has no immediate effect on any asynchronous requests that are currently running (or queued to run) on that server object. Moreover, the length of time these requests have already been running prior to the call does not effect the timer.

The method can be called before or after the asynchronous requests are run. Any asynchronous requests that are run after the method is called must complete before the timer expires. The time available to complete such a request depends on when it was executed, with respect to the CANCEL-REQUESTS-AFTER( ) method call.

The timer has no effect on requests run synchronously during the time interval. Synchronous requests are not cancelled, even if they run longer than the specified timeout.

Manually calling the CANCEL-REQUESTS-AFTER( ) method in the application program during the timeout period has no effect on the timer.

See also

CANCEL-REQUESTS( ) method

CANCELLED attribute

Indicates if the asynchronous request was cancelled using either the CANCEL-REQUESTS( ) method or the DISCONNECT( ) method on the associated server handle.

Data type: LOGICAL

Access: Read-only

Applies to: Asynchronous request object handle

If set to TRUE, the request is cancelled. Otherwise, it is pending or complete (see the COMPLETE attribute).

CAN-CREATE attribute

Indicates whether the ABL user has permission to insert into the database the record associated with a buffer.

Data type: LOGICAL

Access: Read-only

Applies to: Buffer object handle

For information about checking permissions at compile time and run time, see *Manage ABL Applications*.

CAN-DELETE attribute

Indicates whether the ABL user has permission to delete from the database the record associated with a buffer.

Data type: LOGICAL

Access: Read-only

Applies to: Buffer object handle

For information about checking permissions at compile time and run time, see *Manage ABL Applications*.

CAN-DO-DOMAIN-SUPPORT attribute

Indicates whether or not the CAN-DO function treats "@" as the domain delimiter in a fully qualified user ID or as a regular character.

Data type: LOGICAL

Access: Readable/Writable

Applies to: SECURITY-POLICY system handle

If the CAN-DO-DOMAIN-SUPPORT attribute is TRUE, the CAN-DO function treats "@" as a domain delimiter in a fully qualified user ID, as described in the main entry for the CAN-DO function. If the attribute is FALSE, the CAN-DO function treats "@" as a regular character. For example:

- When the attribute is set to TRUE, `CAN-DO("abc","abc@")` evaluates to TRUE because both strings are interpreted as user `abc` in the blank domain.
- When the attribute is set to FALSE, `CAN-DO("abc","abc@")` evaluates to FALSE.

The behavior of the CAN-DO function regarding "@" can be changed at any time in a session by setting the CAN-DO-DOMAIN-SUPPORT attribute appropriately. The attribute defaults to TRUE unless the `-nocandodomain` startup parameter is used, in which case the attribute defaults to FALSE. For more information on the `-nocandodomain` startup parameter, see the *Startup Command and Parameter Reference*.

CAN-READ attribute

Indicates whether the ABL user has permission to read the record associated with a buffer or buffer-field.

Data type: LOGICAL

Access: Read-only

Applies to: Buffer object handle, Buffer-field object handle

For information about checking permissions at compile time and run time, see *Manage ABL Applications*.

CAN-WRITE attribute

Indicates whether the ABL user has permission to modify the record associated with a buffer or buffer-field.

Data type: LOGICAL

Access: Read-only

Applies to: Buffer object handle, Buffer-field object handle

For information about checking permissions at compile time and run time, see *Manage ABL Applications*.

CAREFUL-PAINT attribute

Indicates whether overlapping widgets in a 3D frame will refresh (repaint) carefully but more slowly (TRUE), or quickly, but possibly not as carefully (FALSE).

The CarefulPaint setting in the Startup section of the `progress.ini` file is used to determine the initial setting of the CAREFUL-PAINT attribute. The default value is TRUE.

You can set this frame attribute at any time.

Data type: LOGICAL

Access: Readable/Writable

Applies to: FRAME widget

CASE-SENSITIVE attribute

Indicates whether a buffer-field is case-sensitive.

Data type: LOGICAL

Access: Read-only

Applies to: Buffer-field object handle

CENTERED attribute

Indicates whether the AVM automatically centers the frame in a window.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget

The default value for this attribute is FALSE. When you set this attribute from FALSE to TRUE, the values of the COLUMN, ROW, X, and Y attributes for the frame change to reflect the new location of the frame. Setting the CENTERED attribute from TRUE to FALSE has no meaning and results in an error message.

CHARSET attribute

The current setting of the Character Set (`-charset`) parameter.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

The CHARSET attribute returns a value that specifies the character set used for ABL data, which is either "`iso8859-1`" or "`undefined`". The value is set by the Character Set (`-charset`) parameter.

This attribute is obsolete. See the CPINTERNAL attribute.

CHECKED attribute

The display state for a toggle box or a toggle-box menu item.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: MENU-ITEM widget, TOGGLE-BOX widget

When this attribute is TRUE, the center of the toggle is filled to represent the "on" state of the value associated with the widget. Setting this attribute to FALSE removes the fill from the center of the toggle to represent the "off" state for the value associated with the widget.

CHILD-BUFFER attribute

Returns the buffer handle of the child member of the data-relation object.

Data type: HANDLE

Access: Read-only

Applies to: Data-relation object handle

CHILD-NUM attribute

Returns the relative number assigned to this XML node among its siblings. XML nodes that have the same parent are called siblings, and are numbered from 1 to the number of siblings.

Data type: INTEGER

Access: Read-only

Applies to: X-noderef object handle

The Unknown value (?) is returned if the node reference does not refer to an element node, or if the node is an XML document node.

The following example demonstrates the use of the CHILD-NUM attribute:

```
my-index = hNoderef:CHILD-NUM.
```

CLASS-TYPE attribute

Returns the class type of the most recently compiled class definition (`.cls`) file. If the most recently compiled file was not a class definition file, this attribute returns the empty string ("").

Data type: CHARACTER

Access: Read-only

Applies to: COMPILER system handle

CLEAR( ) method (Call Object Handle)

Resets each attribute of a call object to its default value, which lets you reuse a call object.

Return type: LOGICAL

Applies to: Call object handle

Syntax

```
CLEAR ( )
```

This method resets each attribute of a call object to its default value, which lets you reuse a call object. To reset just the parameters, set the NUM-PARAMETERS attribute to zero.

Note: The return value of this method is not meaningful and should not be relied on.

CLEAR( ) method (DEBUGGER Handle)

Initializes the internal state of the Application Debugger.

Return type: LOGICAL

Applies to: DEBUGGER system handle

Syntax

```
CLEAR ( )
```

This method has no visible effect on the Debugger window. If the Debugger is initialized, this method returns TRUE. Otherwise, it returns FALSE with no effect.

Note: To use this method with the DEBUGGER system handle, you must have the Application Debugger installed in your OpenEdge environment.

For more information on using this method for the Debugger, see the reference entry for the DEBUGGER system handle and *Troubleshoot ABL Applications*.

CLEAR( ) method (FILL-IN Widget)

Clears a fill-in widget's data.

Return type: LOGICAL

Applies to: FILL-IN widget

Syntax

```
CLEAR ( )
```

This method clears a static or dynamic fill-in widget's data, but unlike the CLEAR statement, it is applied to a single fill-in, and it does not clear the fill-in's colors. Using this method on a frame or a field-level widget other than a fill-in raises an error. The method can be used in GUI or character mode.

Note: The return value of this method is not meaningful and should not be relied on.

CLEAR( ) method (ProDataSet Object Handle)

Removes all elements from a dynamic ProDataSet object including buffers and relations.

Return type: LOGICAL

Applies to: ProDataSet object handle

Syntax

```
CLEAR ( )
```

This method restores the state of the dynamic ProDataSet object to what it was immediately after the CREATE DATASET statement.

If an error occurs while executing the CLEAR( ) method, the method returns FALSE. Otherwise, it returns TRUE.

Note: If you call CLEAR( ) or CREATE-LIKE( ) more than once on a dataset, the dynamic temp-tables from the prior call(s) are not deleted automatically and have the potential to cause a memory leak. Therefore the application is responsible for deleting the temp-tables if CLEAR( ) or CREATE-LIKE( ) are called again.

CLEAR( ) method (Temp-Table Object Handle)

Clears temp-table definitions and removes temp-table data.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
CLEAR ( )
```

This method empties the temp-table and removes all its definitional data (field and index definitions and pending saved data). This puts the temp-table object into the CLEAR state, as opposed to the UNPREPARED or PREPARED state. Calling any method after this one changes the state to UNPREPARED.

If an error occurs while executing the CLEAR( ) method, the method returns FALSE. Otherwise, it returns TRUE.

CLEAR-APPL-CONTEXT( ) method

Clears the application context for the current session.

Return type: LOGICAL

Applies to: AUDIT-CONTROL system handle

Syntax

```
CLEAR-APPL-CONTEXT( )
```

After calling this method, the APPL-CONTEXT-ID attribute is cleared for all connected audit-enabled databases and is no longer recorded in audit event records for this application context.

There can be only one active application context per session at any one point in time. To set a different application context for the session, you can:

- Call the CLEAR-APPL-CONTEXT( ) method, to clear the current application context, and then call the SET-APPL-CONTEXT( ) method with the new application context.
- Call the SET-APPL-CONTEXT( ) method with the new application context. If there is an existing application context in effect, the AVM clears the existing application context before setting the new application context.

Calling this method does not generate an audit event or an audit record.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

See also

APPL-CONTEXT-ID attribute, SET-APPL-CONTEXT( ) method

CLEAR-LOG( ) method

Clears all messages existing in the current client log file and leaves the file open for writing.

Note: This method is valid only for DataServers, interactive clients, and batch clients. WebSpeed agents and application servers write a message to the server log file indicating that it is invalid to use the CLEAR-LOG( ) method to clear a WebSpeed or application server log file. In this case, the method returns FALSE.

Return type: LOGICAL

Applies to: DSLOG-MANAGER system handle, LOG-MANAGER system handle

Syntax

```
CLEAR-LOG( )
```

If the CLEAR-LOG( ) method successfully clears the open log file, it returns TRUE.

If the CLEAR-LOG( ) method fails, it returns FALSE and displays a warning message indicating the reason for the failure.

If there is no client log file, the CLEAR-LOG( ) method returns FALSE and displays a warning message that the operation is not valid when there is no log file.

If you specified a log file threshold with the Log Threshold (`-logthreshold`) startup parameter, the CLEAR-LOG( ) method deletes any existing log files that match the name of the LOGFILE-NAME attribute, the Client Logging (`-clientlog`) startup parameter, or the DataServer Logging (`-dslog`) startup parameter. The method then re-creates and opens the first log file in the sequence and changes the LOGFILE-NAME attribute to reflect this.

CLEAR-SELECTION( ) method

Removes the highlight from the currently selected text.

Return type: LOGICAL

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

Syntax

```
CLEAR-SELECTION ( )
```

If the highlight is removed, the method returns TRUE. Otherwise, it returns FALSE.

For browses, this method applies only to fill-in and combo-box browse columns.

CLEAR-SORT-ARROWS( ) method

Clears sort-arrow indicators for all columns in a browse.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
CLEAR-SORT-ARROWS ( )
```

This method always returns TRUE.

See also

SET-SORT-ARROW( ) method

CLIENT-CONNECTION-ID attribute

For a session-managed application, this attribute returns the connection ID for the physical application server connection associated with this server handle.

For a session-free application, because there are is no single physical connection between the client and application server, the value of this attribute has no meaning.

For Web services, this attribute returns the empty string.

Data type: CHARACTER

Access: Read-only

Applies to: Server object handle

This value, assigned by the application server, accepts a connection request from a client application. The connection ID is used as an identifier when any information associated with the connection is logged.

The same connection ID is available to an ABL client application using the CLIENT-CONNECTION-ID attribute and to the application server session servicing the client on the same connection using the SERVER-CONNECTION-ID attribute on the SESSION handle.

The value of the connection ID is guaranteed to be globally unique for all time within a single computer network. Connection IDs can be compared to each other strictly for equality, but other types of comparisons are irrelevant.

For a client, the connection ID of the associated application server connection remains the same until the client disconnects from the application server. If the client reconnects to the same application server, the connection ID of the new connection (and thus the value of the CLIENT-CONNECTION-ID attribute for that connection) is different from the connection ID of the previous connection.

Note: You can identify client context for a user login session of a session-managed or session-free application using the ClientContextId property of the Progress.Lang.OERequestInfo class.

CLIENT-TTY attribute

Returns the name of the terminal display for this user's login session. If not specified, the AVM returns a zero-length character string.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

Once the client-principal object is sealed, this attribute is read-only.

See also

CLIENT-TTY attribute

CLIENT-TYPE attribute

Returns the type of ABL client currently executing.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

The following table shows the value of CLIENT-TYPE for each supported client type.

Table 67: CLIENT-TYPE attribute values

Type of client	Attribute value
ProVision standard ABL client	4GLCLIENT
WebClient	WEBCLIENT
AppServer agent	APPSERVER
WebSpeed agent	WEBSPEED
Progress Application Server agent	MULTI-SESSION-AGENT
Other special-purpose clients	Unknown value (?)

CLIENT-WORKSTATION attribute

The name of the host workstation on which the user, represented by the client-principal object, is working. If not specified, the AVM returns a zero-length character string.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

Once the client-principal object is sealed, this attribute is read-only.

See also

CLIENT-TTY attribute

CLONE-NODE( ) method

Clone the XML node referred to by a node reference. The first parameter must be a valid X-noderef handle and refers to the new cloned XML node if the method succeeds. The new node is associated with the same document, but needs to be inserted with INSERT-BEFORE( ) or APPEND-CHILD( ) to become part of the document structure.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
CLONE-NODE ( x-node-handle , deep )
```

x-node-handle

A valid X-noderef handle to use for the new XML node.

deep

A logical that if TRUE specifies that the whole sub-tree is to be cloned. The default value is FALSE.

The following example demonstrates the use of the CLONE-NODE( ) method to clone an entire sub-tree:

```
hOldNode:CLONE-NODE(hNewNode,true).
```

CLOSE-LOG( ) method

Closes the current log file, which stops an interactive or batch client from writing messages to the log file.

The CLOSE-LOG( ) method writes a message to the log file indicating that the client intentionally closed the log file, so that the user knows why there are no more messages in the log file.

Return type: LOGICAL

Applies to: LOG-MANAGER system handle

Syntax

```
CLOSE-LOG( )
```

If the CLOSE-LOG( ) method successfully closes the open log file, it returns TRUE.

If the CLOSE-LOG( ) method fails, it returns FALSE and displays a warning message indicating the reason for the failure.

If there is no client log file, the CLOSE-LOG( ) method returns TRUE and does not display a warning message.

Note: This method is valid only for interactive and batch clients. Application servers write a message to the server log file indicating that it is invalid to use the CLOSE-LOG( ) method to close an application server log file. In this case, the method returns FALSE. Unlike the other LOG-MANAGER methods, you cannot use this method with the DSLOG-MANAGER system handle.

CODE attribute

A numeric code associated with the last event.

Data type: INTEGER

Access: Read-only

Applies to: LAST-EVENT system handle

For keyboard and mouse events (EVENT-TYPE attribute set to "KEYPRESS" or "MOUSE"), this is the key code. For high-level ABL events (EVENT-TYPE attribute set to "PROGRESS"), this is a unique numeric value greater than the key code values. For information on key codes, see the topics on handling user input in *OpenEdge Programming Interfaces*.

If a mouse event is high-level mouse event (for example, MOUSE-SELECT-CLICK), this attribute is set to the key code of the low-level mouse event (for example, MOUSE-SELECT-UP) that triggered the high-level event. To determine the triggered high-level event, you must also check the value of the FUNCTION attribute, in this case "MOUSE-SELECT-CLICK".

CODEPAGE attribute

The code page of specified r-code.

Data type: CHARACTER

Access: Read-only

Applies to: RCODE-INFO handle

This attribute references the code page of the strings in the text segment. The code page value is written to the r-code file when the file is saved. ABL uses the code page specified by the R-code Out Code Page (`-cprcodeout`) startup parameter to write the r-code text segment. If -cprcodeout is not specified, ABL uses the value of the Internal Code Page (CPINTERNAL) SESSION handle.

For a file that is session compiled, the return value is the Unknown value (?).

COLUMN attribute

The column position of the left edge of the widget or the column position of the mouse cursor for the last mouse event on the display.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LAST-EVENT system handle, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

For browse cells, field groups, and the LAST-EVENT handle, it is readable only.

For all widgets except windows, the COLUMN attribute specifies the location, in character units, of the left edge of the widget relative to the left edge of its parent widget. In windows, the location is relative to the left edge of the display.

For browse columns, the COLUMN attribute returns the Unknown value (?) if the column is hidden.

For control-frames, the COLUMN attribute maps to the Left property of the control-frame COM object (ActiveX control container).

For the LAST-EVENT handle, the COLUMN attribute specifies the column location, in character units, of the last mouse event relative to the left edge of the current frame.

This attribute is functionally equivalent to the X attribute.

COLUMN-BGCOLOR attribute

The color number of the background color for the columns in a browse widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column)

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

COLUMN-DCOLOR attribute
(Character interfaces only)

The number of the display color of a column.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column)

Overrides the color specified for the entire browse widget to display a single column in the specified color.

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

COLUMN-FGCOLOR attribute

The color number of the foreground color for the columns in a browse widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column)

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

COLUMN-FONT attribute
(Graphical interfaces only)

The font for the columns in a browse widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column)

The font number represents an entry in the font table maintained by the FONT-TABLE handle.

COLUMN-LABEL attribute

A text string that describes a column of data associated with a buffer-field.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer-field object handle

COLUMN-MOVABLE attribute
(Graphical interfaces only)

Indicates whether you can move a browse column by pointing, clicking, and dragging.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget

If COLUMN-MOVABLE is TRUE, you can move any of the browse's browse columns by pointing, clicking, and dragging on a column label. This attribute lets you turn on and off the ability of end users to move browse columns. IF COLUMN-MOVABLE is TRUE, the START-MOVE and END-MOVE events for the browse column take precedence over all other events. That is, while this attribute is TRUE, searching on columns is suspended, and the AVM interprets the MOUSE-DOWN event as the column move.

The COLUMN-MOVABLE attribute has no effect on the MOVE-COLUMN( ) method, which lets you move a browse column programmatically.

For updateable browses, if a browse column moves, its tab order changes.

When you set a browse's COLUMN-MOVABLE attribute to a certain value, the AVM automatically sets the MOVABLE attribute of the browse's browse columns to the same value. For more information on the MOVABLE attribute, see the MOVABLE attribute reference entry.

COLUMN-PFCOLOR attribute
(Character interfaces only)

The color number for the display color of a column with input focus.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column)

Overrides the color specified for the entire browse widget to display a single column in the specified color.

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

COLUMN-READ-ONLY attribute

Indicates whether you can tab to a browse column but not edit it.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget (column)

For combo-box browse columns, this attribute is read-only and always returns FALSE.

Note: The COLUMN-READ-ONLY attribute of the browse column corresponds to the READ-ONLY attribute of the fill-in, while the READ-ONLY attribute of the browse column corresponds to the SENSITIVE attribute of the fill-in.

COLUMN-RESIZABLE attribute
(Graphical interfaces only)

Indicates whether you can resize a browse column by pointing, clicking, and dragging.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

If COLUMN-RESIZABLE is TRUE, you can resize any of the browse's browse columns by pointing, clicking, and dragging on a column separator. This attribute lets you turn on and off the ability of end users to resize browse columns. IF COLUMN-RESIZABLE is TRUE, the START-RESIZE and END-RESIZE events for the browse column take precedence over all other events.

When you set a browse's COLUMN-RESIZABLE attribute to a certain value, the AVM automatically sets the RESIZABLE attribute of the browse's browse columns to the same value. For more information on the RESIZABLE attribute, see the RESIZABLE attribute reference entry.

If you add the ResizeinCell option to the [Startup] section of the `progress.ini` file and set it to NO, you limit the user's ability to change the browse column size to only at the column label cell's separator. If a column does not have a label, the user cannot change the size of that column.

If the ResizeinCell option is set to YES, you get the default behavior where the user can change the column size at any place on the column separator.

COLUMN-SCROLLING attribute
(Windows only)

The horizontal scrolling behavior of a browse widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

If the COLUMN-SCROLLING attribute is TRUE, horizontal scrolling for the browse widget moves in whole-column increments. If a column is wider than the browse widget, you cannot see the text if the right side of the column, but you can still scroll to the next column.

If the attribute is FALSE, horizontal scrolling for the browse widget moves in increments equal to the pixel width of the average character in the current browse font. In this case, if a column is wider than the browse, you can view it by scrolling through the column in these increments.

The default value is TRUE for a static browse, and FALSE for a dynamic browse.

COM-HANDLE attribute
(Windows only)

The component handle to the control-frame COM object.

Data type: COM-HANDLE

Access: Read-only

Applies to: CONTROL-FRAME widget

This handle provides access to the ActiveX control container (COM object) associated with the control-frame. You can use this, in turn, to access control-frame COM object properties and methods.

For information on the properties and methods on the control-frame COM object, see the CONTROL-FRAME widget reference entry. For information on accessing COM object properties and methods, see Accessing COM object properties and methods.

COMPLETE attribute

Indicates if the asynchronous request is completed and its result is processed on the client.

Data type: LOGICAL

Access: Read-only

Applies to: Asynchronous request object handle

If set to FALSE, the PROCEDURE-COMPLETE event on this handle has not yet been returned from the application server running the request. This attribute is set to TRUE when the application server returns the PROCEDURE-COMPLETE event and immediately before any specified event procedure or method executes.

CONFIG-NAME attribute

The WebSpeed service name. This method is called by the `get-config` WebSpeed API function. Intended for internal use only.

Data type: CHARACTER

Access: Read-only

Applies to: WEB-CONTEXT system handle

CONNECT( ) method (AppServer)

Physically connects and associates an AppServer instance, or logically connects an application service, with the specified server handle. The current application becomes a client application of the connected AppServer.

Return type: LOGICAL

Applies to: Server object handle

Syntax

```
CONNECT ( { connection-parms }
          [ , userid ]
          [ , password ]
          [ , app-server-info ] )
```

connection-parms

A character string containing a space-separated list of one or more connection parameters necessary to establish an AppServer connection. These parameters include two types:

- A basic set used to connect to an AppServer instance or application service, regardless of the session model
- A set for specifying and managing the session model of the connection

The following table describes the basic connection parameters you must specify to connect to an AppServer instance or application service, regardless of the session model.

Table 68: AppServer basic connection parameters

Connection parameter[81]	Description
`-AppService` *`application-service`*	If you connect through a NameServer, the name of an Application Service supported by the specified NameServer. (Defaults to the default service for the specified Name Server.) If you connect directly to an AppServer, this parameter is ignored.
`-H` **[** *`host_name`* **|** *`IP-address`* **]**	The network address to a NameServer machine or, if you connect directly, to an AppServer machine. You can specify either the TCP/IP host name or the Internet protocol address of the machine. (Defaults to `localhost`)
`-S` **[** *`service-name`* **|** *`port-number`* **]**	The UDP port number for a NameServer, or, if you connect directly, the TCP/IP port number for an AppServer connection. You can specify either an explicit port number or a service name. If you use a service name, the method uses the port number associated with that name in the TCP/IP services file. (Defaults to `5162`)
`-DirectConnect`	If specified, the `-H` and `-S` parameters are interpreted as the network address and TCP/IP port number of an AppServer connection. Otherwise, the `-H` and `-S` parameters are interpreted as the network address and UDP port number of a NameServer.
`-ssl`	If specified, the connection is direct to the AppServer using Transport Layer Security(TLS) tunneling. (Used in conjunction with the `-AppService`, `-H`, and `-S` parameters). **Note:** Be sure you need TLS before using this option. TLS incurs more or less heavy performance penalties, depending on resources and load.
`-nosessionreuse`	If specified, the connection does not reuse the TLS session ID when reconnecting to the same TLS-enabled server (either a Web server with HTTPS or a TLS-enabled AppServer).
`-nohostverify`	If specified, turns off host verification for a TLS-enabled connection, either using HTTPS with the AIA or using a direct connection to a TLS-enabled AppServer. Without this parameter specified, the client compares the host name specified in the connection with the Common Name specified in the server certificate, and raises an error if they do not match. With this parameter specified, the client never raises the error. For more information, see *Learn about Security and Auditing*.

[81] Previous versions of the AppServer allow you to include a network protocol using the `-N` parameter, which must always specify `TCP`. While still allowed, it is optional and always defaults to TCP/IP.

Connection parameter[81]	Description
`-pf` *`filename`*	A text file containing any of the other AppServer connection parameters described in this table or Table 69: AppServer session model connection parameters. If this file contains any other OpenEdge startup parameters, the method ignores them.
`-AppServerKeepalive` *`capstr`*	Indicates that the client would like to employ the AppServer Keepalive protocol on this connection, if supported and enabled by the AppServer. To enable the protocol, specify the `allowServerASK` value for *`capstr`*. To disable the protocol, specify `denyServerASK`. The default value is `allowServerASK`. The absence of this property indicates that the default value for the ServerASK protocol will be used on this connection.
`-URL` *`Web-or-AppServer-path`*	An HTTP (or HTTPS-based) URL to an AIA (for an Internet-secure AppServer connection) or an AppServer-based URL (with or without TLS tunneling for a TLS-enabled AppServer connection). **Note:** Be sure you need TLS (either, and especially both, an HTTPS or TLS-enabled AppServer) before using this option. TLS at any point in a networked application incurs more or less heavy performance penalties, depending on resources and load.
-clientConnectTimeout *`nMilliseconds`*	The number of milliseconds to wait before timing out each connection attempt.

Note: Connections to an Internet-secure (HTTPS) or TLS-enabled AppServer require the management of public keys on the client (TLS client) and private keys on the server (TLS server). For an Internet-secure AppServer, the TLS server is the Web server that hosts the AIA. For a TLS-enabled AppServer, the TLS server is the AppServer itself. For information on configuring a Web server for HTTPS, see your Web server documentation. For information on managing private key and digital certificate stores for TLS clients and servers, see *Learn about Security and Auditing*.

The following table describes connection parameters for specifying and managing the session model of the connection.

Table 69: AppServer session model connection parameters

Connection parameter	Session model/ default	Description
`-sessionModel` *`sessionModel`*	Session-managed Session-free	Session model supported by the AppServer operating mode, specified by one of the following values: • `Session-managed` • `Session-free` This value is **not** case sensitive. This parameter is required for session-free applications and is optional for session-managed applications. This value must match the AppServer operating mode or the `CONNECT( )` method fails. The default value is `Session-managed.`
`-connectionLifetime` *`nSeconds`*	Session-free	The maximum number of seconds that a given connection can be used before it is destroyed. Connections whose lifetime exceeds the specified value are destroyed as they become available. An available connection is one that is not currently reserved to run a request. Bound connections associated with remote persistent procedures are not available for re-use until the persistent procedure is deleted. So, bound connections remain available as long as necessary, even if they exceed the specified value. The default value is 300 seconds.
`-initialConnections` *`nConnections`*	Session-free	The number of connections established when the `CONNECT( )` method executes on a given server handle. The value must be greater than zero. If the specified number of connections cannot be created, the `CONNECT( )` method fails and any successfully-created connections are closed. The default value is 1.

Connection parameter	Session model/ default	Description
`-maxConnections` *nConnections*	Session-free	The maximum number of connections that can be created for a given server handle to execute non-persistent external procedures. The value must be greater than or equal to zero. If this value is zero, there is no limit to the number of connections that can be created. **Note:** For calls to persistent procedures, their internal procedures, and user-defined functions, the client has no limit on the number of connections that can be created. The default value is 0.
`-nsClientMaxPort` *portNum*	Session-managed Session-free	The maximum value for the UDP port number used by the client when communicating with the NameServer. If this value is zero, the AVM chooses the NameServer client port randomly. This value should be greater than or equal to the value of the `-nsClientMinPort` parameter. The default value is 0.
`-nsClientMinPort` *portNum*	Session-managed Session-free	The minimum value for the UDP port number used by the client when communicating with the NameServer. If this value is zero, the AVM chooses the NameServer client port randomly. The default value is 0.
`-nsClientPicklistExpiration` *nSeconds*	Session-free	The maximum amount of time, in seconds, that the client retains an AppServer pick list for an application service. The default value is 300.
`-nsClientPicklistSize` *nPicks*	Session-free	The number of AppServer picks to request from the NameServer each time it looks up the available AppServer connections for a given application service name. The default value is 1.

Connection parameter	Session model/ default	Description
`-nsClientPortRetry` *nRetries*	Session-managed Session-free	The maximum number of attempts that the client makes to get a valid local UDP port number when attempting to communicate with the NameServer. The default value is 0.
`-nsClientDelay` *nMilliSeconds*	Session-managed Session-free	The interval, in milliseconds, that the client waits between attempts to get a valid UDP port number when attempting to communicate with the NameServer. The default value is 0.

Note that the actual AppServer that the client connects to is controlled by the NameServer based on the application service (`-AppService`) name specified by the client. The ABL interface in cooperation with the NameServer connect the client application to one of the AppServer instances that supports the specified application service. If you do not specify an application service, the NameServer uses whatever AppServer registers itself as the default service, if any.

If the application service is unknown to the NameServer, the client application receives an error. Otherwise, the connection proceeds and any configured Connect procedure executes for the connected AppServer.

[*userid* **][** , *password***][** , *app-server-info***]**

From one to three character string parameters passed as input to the AppServer Connect procedure. The possible values that you can specify for these parameters are determined by the Connect procedure for the AppServer application. If you omit a parameter, it defaults to the Unknown value (?).

If an error occurs while executing the `CONNECT( )` method, the method returns FALSE. Otherwise, it returns TRUE. An error can occur if:

- The server handle is invalid.
- One of the parameters contains an invalid value.
- One of the values specified in the *connection-parms* parameter is invalid.
- The Name Server cannot be located.
- The specified Application Service is not registered to a NameServer.
- The client application cannot connect to the AppServer selected by the NameServer.
- The AppServer selected by the NameServer cannot allocate a connection for the client application.
- The AppServer executes a Connect procedure that terminates with a `STOP` condition, a `QUIT` condition, or after executing a `RETURN ERROR` statement.

If the `CONNECT( )` method completes successfully, the CONNECTED( ) method returns TRUE.

The connection lasts until the client application executes the server handle `DISCONNECT( )` method or until the AVM detects any failure conditions that automatically terminate the connection.

The `-URL` connection parameter allows you to connect to an AppServer using the AppServer Internet Adapter (AIA) with the following protocols: HTTP and HTTPS.

CONNECT( ) method (Progress Application Server for OpenEdge)

Connects and associates an instance of the Progress Application Server (PAS) for OpenEdge with the specified server handle and defines the application session model for the connection. This allows a client program to run remote procedures on a PAS for OpenEdge instance. The CONNECT( ) method requires the host and port to connect to and the application server transport value (apsv) in a URL format. Optionally you can specify a web application name, userid, password, and other connection parameters. The CONNECT( ) method returns TRUE if it completes successfully.

Return type: LOGICAL

Applies to: Server object handle

Syntax

```
CONNECT ( {connection-parms} [,userid] [,password] [,app-server-info] )
```

connection-parms

A character string containing a space-separated list of one or more connection parameters necessary to establish a connection to the instance.

Table 70: PAS for OpenEdge connection parameters

Connection parameter	Required	Description
-URL *scheme://host:port/*[*web-app-name/*]*transport*	Yes	The URL of the server to which you are connecting, where: *scheme* — Is either HTTP or HTTPS. *host* — Specifies the name or domain of the host. *port* — Specifies the port for the host connection that the instance is running on. *web-app-name* — Specifies the OpenEdge ABL web application to connect with. If *web-app-name* is not specified, a connection to the default OpenEdge ABL web application (ROOT) is used. *transport* — Specifies the ABL client (or Open Client) transport. Must be **apsv**.
-nosessionreuse	No	Does not reuse the TLS session ID when reconnecting over HTTPS to the same instance.
-nohostverify	No	Turns off host verification for an HTTPS connection. If *host* and the Common Name from the server certificate do not match, no error is raised and the secure encrypted connection is still made if all other certificate information is correct. If the parameter is not specified and *host* does not match the server certificate, an error is raised. See *Learn about Security and Auditing* for more information.

Connection parameter	Required	Description
-pf *filename*	No	Specifies a parameter file containing additional connection parameters. The parameters allowed are described in Table 69: AppServer session model connection parameters. If *filename* contains other OpenEdge startup parameters, they are ignored by this method
-sessionModel *option*	No	Specifies the application session model required by the ABL client for this connection. It can be one of two options: • `Session-managed` • `Session-free` This value is **not** case sensitive. `Session-managed` is the default.
-connectionLifetime *nSeconds*	No	The maximum number of seconds that a given connection can be used before it is destroyed. Connections whose lifetime exceeds the specified value are destroyed as they become available. An available connection is one that is not currently reserved to run a request. Bound connections associated with remote persistent procedures are not available for re-use until the persistent procedure is deleted. So, bound connections remain available as long as necessary, even if they exceed the specified value. The default value is 300 seconds. This parameter is only applicable if `-sessionModel session-free` is used.
-initialConnections *nConnections*	No	The number of connections established when the `CONNECT( )` method executes on a given server handle. The value must be greater than zero. If the specified number of connections cannot be created, the `CONNECT( )` method fails and any successfully-created connections are closed. The default value is 1. This parameter is only applicable if `-sessionModel session-free` is used.

Connection parameter	Required	Description
-maxConnections *nConnections*	No	The maximum number of connections that can be created for a given server handle to execute non-persistent external procedures. The value must be greater than or equal to zero. If this value is zero, there is no limit to the number of connections that can be created. **Note:** For calls to persistent procedures, their internal procedures, and user-defined functions, the client has no limit on the number of connections that can be created. The default value is 0. This parameter is only applicable if `-sessionModel session-free` is used.
-clientConnectTimeout *nMilliseconds*	No	The number of milliseconds to wait before timing out each connection attempt.

The client determines the application session model that the instance uses for each connection, because ABL sessions run using any application model that a given client requires.

Connections to an Internet-secure (HTTPS) instance require the management of digital certificate stores for public keys on the client (TLS client) and private keys on an instance (TLS server). For information on configuring for HTTPS connections, see the *Manage Progress Application Server for OpenEdge*. For information on using TLS to secure the client side of a PAS for OpenEdge application, see the *Progress Application Server for OpenEdge: Application Migration and Development Guide*. For information on managing digital certificate stores for OpenEdge TLS clients, see *Learn about Security and Auditing*.

userid

Optional string parameter containing the userid to connect with. It is passed as input to the instance's `Connect` procedure.

password

Optional string parameter containing the password to connect with. It is passed as input to the instance's `Connect` procedure.

app-server-info

Optional string parameter containing application server information. It is passed as input to the instance's `Connect` procedure.

Possible values for *userid*, *password*, and *app-server-info* are determined by the `Connect` procedure for the business application. The `Connect` procedure is specified by the `sessionConnectProc` property in the instance's `openedge.properties` file. If any of these parameters are omitted, the Unknown value (?) is passed.

Example

The following is an example of creating a session-free connection to an OpenEdge ABL web application named `GeneralLedger` on a local instance:

```
DEFINE VARIABLE happsrv AS HANDLE NO-UNDO.

CREATE SERVER happsrv.
IF happsrv:CONNECT("-URL http://localhost:8810/GeneralLedger/apsv
                    -sessionModel session-free") THEN
DO:
    /* Some accounting work on the local PAS for OpenEdge instance */
END.
```

Notes

If an error occurs while executing the `CONNECT( )` method, the method returns FALSE. Some conditions for which an error can occur include:

- The server handle is invalid.
- One of the parameters contains an invalid value.
- One of the values specified in the *`connection-parms`* parameter is invalid.
- The instance executes a `Connect` procedure that terminates with a `STOP` condition, a `QUIT` condition, or after executing a `RETURN ERROR` statement.

The connection lasts until the client application executes the server handle `DISCONNECT( )` method or until the server session detects any failure conditions that automatically terminate the connection.

For more information on PAS for OpenEdge Connect procedures, see the *Progress Application Server for OpenEdge: Application Migration and Development Guide*

.

CONNECT( ) method (Socket object)

Connects a socket to the specified TCP/IP port on the specified host.

Return type: LOGICAL

Applies to: Socket object handle

Syntax

```
CONNECT ( [ connection-parms ] )
```

`connection-parms`

A character string expression that contains a space-separated list of one or more socket connection parameters.

The following table describes each socket connection parameter, which can be included in this string.

Table 71: Socket connection parameters

Parameter	Description
`-H` *`socket-address`*	Optional. The host name or IP address to which the connection is to be established.
`-nohostverify`	If specified, turns off host verification for a TLS-enabled connection to a server socket. Without this parameter specified, the client compares the host name specified in the connection with the Common Name specified in the server certificate, and raises an error if they do not match. With this parameter specified, the client never raises the error. For more information, see *Learn about Security and Auditing*.
`-nosessionreuse`	If specified, the connection does not reuse the TLS session ID when reconnecting to the same TLS-enabled server socket. **Note:** OpenEdge TLS turns on TLS session reuse by default. So, after the initial connection to a given host (`-H`) and port (`-S`), each **subsequent** connection to the **same** host and port restarts the TLS session and ignores any different connection parameters that are specified for the subsequent connection, including `-nosessionreuse`. If you want to change TLS socket options (such as `-nohostverify`) for each subsequent connection to a given host and port, be sure to specify the `-nosessionreuse` parameter on the **initial** TLS socket connection to that same host and port.
`-pf` *`filename`*	Optional. A text file containing any of the socket connection parameters described in this table. If this file contains any other OpenEdge startup parameters, this method ignores them.
`-S` *`socket-port`*	The port number for the socket connection. You can specify either an explicit port number or a TCP service name. If you use a TCP service name, the method uses the port number associated with that name in the TCP/IP services file.
`-ssl`	If specified, the connection to the server socket uses Transport Layer Security (TLS) tunneling. (Used in conjunction with the `-H` and `-S` parameters.) **Note:** Be sure you need TLS before using this option. TLS incurs more or less heavy performance penalties, depending on resources and load.

Parameter	Description
`-sslciphers` *`SSL-protocol-ciphers`*	Optional. A comma-separated list of cryptographic ciphers the client is willing to accept.
`-sslprotocols` *`SSL-protocol-names`*	Optional. A comma-separated list of cryptographic protocols the client is willing to accept.
`-SSLAuth` *`Authentication-type`*	Specifies if access to an OpenEdge and Non-OpenEdge server requires TLS client authentication. The two authentication types are: **1.** `ssl`—Enables client authentication for OpenEdge and Non-OpenEdge server access. **2.** `basic`— In this authentication type, the connect method ignores the client authentication for OpenEdge and Non-OpenEdge server access.
`-SSLKeyFile` *`filename`* | *`target-database`*	The location of the client certificate. If you do not specify an absolute path of the client certificate file, the connection operation searches the `$DLC/keys` folder for the `client-certificate-file-name.pem` file.This option must be set only if `-SSLAuth` parameter is set to `ssl`.
`-SSLKeyPwd` *`password`*	The TLS client certificate password in clear text or in encoded format. For more information on encoded format, see 'Use genpassword to obtain a keystore password-encrypted value' in *Manage OpenEdge Keys and Certificates*.

Note: Connections to a TLS-enabled server socket require the management of public keys on the client (TLS client) and private keys on the server (TLS server). For ABL sockets, the TLS client is the ABL session initiating the TLS connection on a socket object and the TLS server is the ABL session enabling TLS connections on a server socket object. For information on using TLS to secure an ABL socket connection, see the sections on sockets in *OpenEdge Programming Interfaces*. For more information on TLS and managing private key and digital certificate stores for OpenEdge TLS clients and servers, see *Learn about Security and Auditing*.

If an error occurs while executing the CONNECT( ) method, the method returns FALSE. Otherwise, it returns TRUE.

When an ABL client (that is not TLS-enabled) calls the CONNECT( ) method and immediately reads data from the socket using the READ( ) method, and an ABL server (that is TLS-enabled) calls the ENABLE-CONNECTIONS( ) method and immediately writes data to the socket using the WRITE( ) method, a deadlock condition can occur. That is, the client is waiting for the server to send data, and the server (regardless of the Write operation) is waiting for the client connection to send data that starts the TLS connection.

CONNECT( ) method (Web service)

Connects to and associates a Web service instance with the specified server handle. The current application becomes a client application of the connected Web service.

Return type: LOGICAL

Applies to: Server object handle

Syntax

```
CONNECT ( [ connection-parms ] )
```

connection-parms

A character string containing a space-separated list of one or more connection parameters.

The following table describes each Web service connection parameter you can include in this string. This method ignores any other strings included in this parameter.

Table 72: Web service connection parameters

Parameter	Description
`-WSDL` *wsdl-document*	The location of the WSDL document. This required parameter is the URL, UNC, or local file pathname to the WSDL file that describes the Web service. The document can be local or remote. The location can optionally contain a user's account name and password to use when connecting to the Web Server. For more information, see -WSDL parameter security options. If the protocol is not part of the wsdl-document's URL, the 'file' protocol is assumed. Additionally, the 'file' can be a relative pathname as it is relative to the current working directory.
`-WSDLUserid` *user-id*	Optional user account name to use in connecting to the Web service that hosts the WSDL document. If `-WSDLUserid` is specified and `-WSDLPassword` is not, the AVM uses a blank password.
`-WSDLPassword` *password*	Optional password to use with `-WSDLUserid`. This attribute is ignored if `-WSDLUserid` is not specified.
`-WSDLAuth` *Authentication-type*	Specifies if access to a WSDL file requires TLS client authentication. Set this to `ssl` to enable client authentication for WSDL access. If you set this to `basic`, the connect method ignores client authentication for WSDL access.

Parameter	Description
-WSDLKeyFile *filename* | *target-database*	The location of the client certificate. If you do not specify an absolute path of the client certificate file, the connection operation searches the $DLC/keys folder for the client-certificate-file-name.pem file. This option must be set only if -WSDLAuth parameter is set to ssl.
-WSDLKeyPwd *password*	The TLS client certificate password in clear text or encoded format.
-SSLAuth *Authentication-type*	Specifies if access to a SOAP service requires TLS client authentication. Set this to ssl to enable client authentication for SOAP access. If you set this to basic, the connect method ignores client authentication for SOAP access.
-SSLKeyFile *filename* | *target-database*	The location of the client certificate. If you do not specify an absolute path of the client certificate file, the connection operation searches the $DLC/keys folder for the client-certificate-file-name.pem file.This option must be set only if -SSLAuth parameter is set to ssl.
-SSLKeyPwd *password*	The TLS client certificate password in clear text or in encoded format. For more information on encoded format, see 'Use genpassword to obtain a keystore password-encrypted value' in *Manage OpenEdge Keys and Certificates*.
-Service *service-name*	The local name of the service element within the WSDL document that the application will use. This field is optional. Many WSDL documents only support one service and this parameter is optional if there is only one (or zero) service elements defined. Used in conjunction with -Port.
-ServiceNamespace *service-namespace*	The namespace of the service element within the WSDL document that the application will use. Most WSDL documents only support one service and this parameter is optional if there is only one service defined. This parameter is used in conjunction with -Service and is ignored if -Service was not specified. This parameter is optional, if the namespace is included in -Service.
-Port *port-name*	The local name of the port element contained within the service element. Used in conjunction with -Service. This parameter is optional if -Service contains only one port.
-Binding *binding-name*	The local name of the binding element contained in the WSDL document. Used in conjunction with -SoapEndpoint. This parameter is optional if the WSDL contains only one binding.

Parameter	Description
`-BindingNamespace` *`binding-namespace`*	The namespace of the binding element within the WSDL document that the application will use. This optional field is needed only if the local binding-name is not unique.
`-SOAPEndpoint` *`URL-endpoint`*	The URL identifying the endpoint for this Web service. Used in conjunction with `-Binding`. It is an error to use this parameter in conjunction with `-Service` or `-Port`.
`-SOAPEndpointUserid` *`user-id`*	Optional user account name to use to connect to a Web service that hosts the Web Server application. If `-SOAPEndpointUserid` is specified and `-SOAPEndpointPassword` is not, the AVM uses a blank password.
`-SOAPEndpointPassword` *`password`*	Optional password to use with the `-SoapEndpointUserid`. This attribute is ignored if `-SoapEndpointUserid` is not specified.
`-TargetNamespace` *`targetNamespace`*	The namespace contained in the WSDL document. This parameter can be used as a version check. The information in this parameter is compared against the `-TargetNamespace` contained in the WSDL document. If they do NOT match the CONNECT ( ) method fails.
`-connectionLifetime` *`nSeconds`*	The maximum number of seconds that a given connection can be reused for asynchronous requests before it is destroyed. Connections whose lifetime exceeds the specified value are destroyed as they become available. An available connection is one that is not currently reserved to run an asynchronous request. The default value is 300 seconds.
`-maxConnections` *`num-connections`*	Maximum number of connections maintained between the client and the Web Server for asynchronous requests. If num connections is less than or equal to 0, the application is requesting no predefined limit on the number of connections. If the client application exceeds the specified number of connections, the asynchronous requests are queued. The default value is 0.
`-nosessionreuse`	If specified, the connection does not reuse the TLS session ID when reconnecting to the same HTTPS-enabled Web server.
`-nohostverify`	If specified, turns off host verification for an HTTPS Web server connection. Without this parameter specified, the client compares the host name specified in the connection with the Common Name specified in the server certificate, and raises an error if they do not match. With this parameter specified, the client never raises the error. For more information, see *Learn about Security and Auditing*.

Parameter	Description
-clientConnectTimeout *nMilliseconds*	The number of milliseconds to wait before timing out each connection attempt.
-pf *filename*	A text file containing any of the other Web service binding parameters described in this table. If this file contains any other OpenEdge startup parameters, this method ignores them.

Note: Connections to an Internet-secure (HTTPS) Web service require the management of public keys on the client (HTTPS client) and private keys on the server (HTTPS server). For an Internet-secure Web service, the HTTPS server is the Web server that hosts the Web service. For information on configuring a Web server for HTTPS, see your Web server documentation. For more information on HTTPS and TLS, and on managing private key and digital certificate stores for OpenEdge TLS clients and servers, see *Learn about Security and Auditing*.

-WSDL parameter security options

The -WSDL parameter can optionally contain a user account name and password to use to connect to a Web Server.

This is the syntax for specifying the parameter using the HTTP and HTTPS protocols, respectively:

```
-WSDL http://[user-id[:password]@]
  web-server-host[:web-server-port]WSDL-path
```

```
-WSDL https://[user-id[:password]@]
  web-server-host[:web-server-port]WSDL-path
```

user-id

User account name to use to connect to a Web service that hosts the WSDL document. If *user-id* is specified and password is not, the AVM uses a blank password.

password

Password to use with the *user-id*. This parameter is ignored if *user-id* is not specified.

web-server-host

TCP/IP host address of the Web Server that hosts the WSDL document.

web-server-port

TCP/IP port address of the Web Server that host the WSDL document. The default port is 80 for HTTP and 443 for HTTPS.

WSDL-path

URL path to the WSDL document for the Web service.

Instead of building the account name and password into the WSDL string, you can specify the account name using the `-WSDLUserid` parameter and the password using the `-WSDLPassword` parameter. If these parameters are used and the WSDL URL (*`WSDL-path`*) also contains a user id and password, the information on the WSDL URL is used.

Connection parameter combinations

The CONNECT( ) method is used to connect an ABL SERVER object to a specific application service. This service can be either an AppServer or a Web service. Independent of the type of application service to which the client is connecting, the client needs to provide the location of the service and transport information. There are two mechanisms for providing this information when connecting to a Web service:

1. The CONNECT( ) method can identify a specific service element name and port element name from the WSDL document. The combination of these two element names identify the location of a set of operations that are available on the Web service. It also identifies the transport data. The service element name is specified with the `-Service` connection parameter and the port element name is specified with the `-Port` connection parameter.

   If the WSDL document contains several service elements, the CONNECT method must identify which service element the client wants to connect to, via `-Service`. If the WSDL document only identifies one service element, the CONNECT method does not need to contain the service element name. Similarly if the WSDL document (or if the identified service element) only identifies one port element, the CONNECT method does not need to contain the port element name.

   If the application needs to provide account name and password information, it can accomplish this by providing the account name and password information in the `-SoapEndpointUserid` and `-SoapEndpointPassword` parameters.

   If the WSDL document identifies multiple service elements with the same local name, the CONNECT ( ) method must also contain the `-ServiceNamespace` connection parameter.

2. If the WSDL document contains several binding elements, the CONNECT method must identify which binding element the client wants to use, via the `-Binding` parameter. If the WSDL document only identifies one binding element, the CONNECT method does not need to contain the binding element name.

   If the application needs to provide account name and password information, it can accomplish this by providing the account name and password information in the `-SoapEndpointUserid` and `-SoapEndpointPassword` parameters.

   If the WSDL document identifies multiple binding elements with the same local name, the CONNECT( ) method must also contain the `-BindingNamespace` connection parameter.

If an error occurs while executing the CONNECT( ) method, the method returns FALSE. Otherwise, it returns TRUE. An error can occur if:

- The server handle is invalid.
- One of the parameters contains an invalid value.
- One of the values specified in the *`connection-parms`* parameter is invalid.
- The `-TargetNamespace` does not match the value contained in the WSDL document.
- The WSDL document cannot be located.
- The `-WSDLUserid` or `-WSDLPassword` is not valid.

If the CONNECT( ) method completes successfully, the CONNECTED( ) method returns TRUE.

The connection lasts until the client application executes the server handle DISCONNECT( ) method or until the AVM detects any failure condition that automatically terminates the connection.

CONNECTED( ) method

Indicates whether a Progress Application Server for OpenEdge (PAS for OpenEdge) instance, or Web service is currently connected and associated with the server handle, or if a socket handle is currently connected to a port.

Return type: LOGICAL

Applies to: Server object handle, Socket object handle

Syntax

```
CONNECTED ( )
```

For a Web service or PAS for OpenEdge instance, this method indicates if a server handle is currently connected (that is, if the client has a logical connection to the Web service or PAS for OpenEdge instance). It does not indicate that a physical connection exists between the ABL client and the Web service or PAS for OpenEdge instance.

For a state-reset, state-aware, or stateless application server, this method returns TRUE if the application server is currently connected and associated with the server handle. For a state-free application server, this method returns TRUE if the `CONNECT( )` method has been successfully executed for an application service associated with this handle and at least one application server resource is available for the client to access this application service.

This method returns TRUE between the successful invocation of the `CONNECT( )` method and a call to the `DISCONNECT( )` method. If the connection terminated abnormally (that is, other than by the `DISCONNECT( )` method), the `CONNECTED( )` method returns FALSE.

Note: This method returning TRUE does not indicate the state of the HTTP connection to the Web service or PAS for OpenEdge instance. If there is a failure in the connection between the client and the Web service or PAS for OpenEdge instance, subsequent requests might fail.

However, in OpenEdge 12.3 and above, this method has been improved to return the availability of a valid server session for an ABL client. For more information, see the "Enhanced server object handle CONNECTED() method for ABL clients" entry in What's New in OpenEdge 12.3.

For a socket object, this method returns TRUE if the socket handle refers to a connected socket, and returns FALSE otherwise.

CONTEXT-HELP attribute
(Windows only)

When CONTEXT-HELP is TRUE, a question mark icon displays in the title bar of the window or dialog box. The default value is FALSE. This attribute must be set before the window or dialog box is realized.

Return type: LOGICAL

Access: Readable/Writable

Applies to: DIALOG-BOX widget, WINDOW widget

Due to bugs in Microsoft Windows, the question mark icon does not appear, or appears but does not function, when combined with other attribute settings that affect a window's title bar:

- If CONTEXT-HELP = TRUE and SMALL-TITLE = TRUE, the question mark icon does not appear.
- If CONTEXT-HELP = TRUE and both MIN-BUTTON = TRUE and MAX-BUTTON = TRUE, the question mark icon does not appear.
- If CONTEXT-HELP = TRUE and either (but not both) of MIN-BUTTON or MAX-BUTTON = TRUE, the question mark icon appears but does not function.
- If CONTEXT-HELP = TRUE and CONTROL-BOX = FALSE, the question mark icon does not appear.

To summarize, you must set CONTEXT-HELP = TRUE, MIN-BUTTON = FALSE, and MAX-BUTTON = FALSE (leaving CONTROL-BOX at its default value of TRUE and SMALL-TITLE at its default value of FALSE) in order to successfully use this feature with a window widget.

Note: The preceding settings only apply to window widgets, not to dialog boxes. The question mark icon always functions correctly when used with a dialog box.

CONTEXT-HELP-FILE attribute
(Windows only)

Specifies the path name of a help (`.HLP`) file associated with a dialog box, window, or session.

Return type: CHARACTER

Access: Readable/Writable

Applies to: SESSION system handle, DIALOG-BOX widget, WINDOW widget

If CONTEXT-HELP-FILE is not specified (is unknown) for a dialog box, the dialog box inherits the help file of its parent window. If the parent window's CONTEXT-HELP-FILE is also unknown, it inherits the session's help file (specified by SESSION:CONTEXT-HELP-FILE).

The full pathname of the help file should be given. The AVM does not search for the help file. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

CONTEXT-HELP-ID attribute
(Windows only)

Specifies the identifier of a help topic in a help file.

Return type: INTEGER

Access: Readable/Writable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

CONTROL-BOX attribute
(Windows only)

Indicates whether the window has a system menu box in its caption bar.

Data type: LOGICAL

Access: Readable/Writable

Applies to: WINDOW widget

In character interfaces, this attribute has no effect.

The CONTROL-BOX attribute must be set before the window is realized. The default value is TRUE.

Control-Name property
(Windows only; Graphical interfaces only)

The component handle to an ActiveX control that has the specified design-time name (*Control-Name*) and that is loaded into the control-frame.

Data type: COM-HANDLE

Access: Read-only

Applies to: CONTROL-FRAME widget, COM object

Another way of getting a component handle to an ActiveX control is to access it through the control collection. For more information, see the Controls property entry.

References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see Accessing COM object properties and methods.

To return the component handle of the ActiveX control, you provide the design-time name as a property of the control-frame COM object. In the following example, Spin is the name of an ActiveX control and is also used as a property to return the handle to that control:

```
DEFINE VARIABLE hSpin            AS COM-HANDLE NO-UNDO.
DEFINE VARIABLE hControlFrameCOM AS COM-HANDLE NO-UNDO.
DEFINE VARIABLE hControlFrame    AS HANDLE     NO-UNDO.

/* Instantiate the control-frame with hControlFrame and load the Spin control
   into it*/
hControlFrameCOM = hControlFrame:COM-HANDLE.
hSpin            = hControlFrameCOM:Spin.
```

This is the simplest technique to access an ActiveX control that is loaded in a control frame.

Controls property
(Windows only; Graphical interfaces only)

The component handle to the control collection that references the ActiveX controls in the control-frame.

Data type: COM-HANDLE

Access: Read-only

Applies to: CONTROL-FRAME widget, COM object

Because this release supports only one ActiveX control per control-frame, this control collection references only one control. Once you have the component handle to the control collection (*collection-handle*), you can get the component handle to the ActiveX control itself by invoking the Item( ) method of the control collection. (This is a standard ActiveX convention.) With support limited to a single control per control-frame, the only valid Item( ) method call is *collection-handle*:Item(1). Once you have the component handle to the actual ActiveX control, you can access the properties and methods of that control.

A simpler technique for getting a component handle to an ActiveX control is to reference its name directly as a property of the control-frame COM object. For more information, see the Control-Name property reference entry.

References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see Accessing COM object properties and methods.

CONVERT-3D-COLORS attribute
(Windows only)

Determines whether image colors are converted to the corresponding system 3D colors.

Return type: LOGICAL

Access: Readable/Writable

Applies to: BUTTON widget, IMAGE widget

The default value of this attribute for a button is TRUE; the default value for an image is FALSE.

The CONVERT-3D-COLORS attribute can be set after a widget is realized, but it will not take effect until an image is loaded into the widget using one of the following methods: LOAD-IMAGE( ), LOAD-IMAGE-UP( ), LOAD-IMAGE-DOWN( ), or LOAD-IMAGE-INSENSITIVE( ).

If the CONVERT-3D-COLORS attribute is TRUE either when the widget is realized or when any of the LOAD-IMAGE methods is executed, the AVM will convert the shades of gray in the image after loading it. The following table identifies and describes the colors that are converted.

Table 73: 3D-color conversions

If the color is . . .	And the original Red-Green-Blue (RGB) color value is . . .	Then the new converted system color is . . .
White	(255, 255, 255)	System button highlight color
Light Gray	(192, 192, 192)	System button face color
Dark Gray	(128, 128, 128)	System button shadow color
Black	(0, 0, 0)	System button text color

CONVERT-TO-OFFSET( ) method

Converts a row and column value to a character offset in an editor widget.

Return type: INTEGER

Applies to: EDITOR widget

Syntax

```
CONVERT-TO-OFFSET ( row , column )
```

row

An integer row number.

column

An integer column number.

In Windows, both the regular editor and the large editor support CONVERT-TO-OFFSET.

COPY-DATASET( ) method

Copies a source ProDataSet object to a target ProDataSet object. By default, the AVM empties the target ProDataSet object temp-tables of all records before copying the source ProDataSet object.

Return type: LOGICAL

Applies to: ProDataSet object handle

This is the syntax to invoke this method:

Syntax

```
targ-dataset-handle:COPY-DATASET ( src-dataset-handle [ , append-mode
  [ , replace-mode [ , loose-copy-mode [ , pairs-list [ , current-only
  [ , name-prefix ] ] ] ] ] ] )
```

targ-dataset-handle

The handle to the target ProDataSet object to receive the copy.

src-dataset-handle

The handle to the source ProDataSet object from which to copy temp-table records to the target ProDataSet object on which you execute the method.

append-mode

An optional logical expression where TRUE indicates that the AVM copy the ProDataSet object temp-tables in an *append* mode.

When TRUE, the AVM appends the source ProDataSet temp-tables to the target ProDataSet temp-tables. If there is a unique index on a target temp-table and the AVM finds a row with a duplicate key, it does not replace the row. If there is not a unique index on the target temp-table, the AVM appends the data row from the source temp-table to the target temp-table, which might result in duplicate rows. If this operation results in duplicate rows, the AVM does not generate a run-time error.

Note: In this mode, the AVM treats duplicate temp-table rows during the copy operation in the same way it treats duplicate temp-table rows during a fill operation in MERGE mode.

When you are certain the ProDataSet temp-tables do not contain duplicate rows, copying the ProDataSet object in append mode is more efficient than copying it in replace mode.

When FALSE, the AVM does not append records in the target ProDataSet temp-tables. The default value is FALSE.

The AVM ignores this expression when *replace-mode* is TRUE.

replace-mode

An optional logical expression where TRUE indicates that the AVM copy the ProDataSet object temp-tables in a *replace* mode.

When TRUE, the AVM replaces records in the target ProDataSet object temp-tables with corresponding records from the source ProDataSet temp-tables. The target temp-table must have a unique primary index that the AVM can use to find the corresponding records. When a corresponding record is found in the target temp-table, the AVM replaces the target record with the source record. When a corresponding record is not found in the target temp-table, the AVM creates a new target record using the source record. If the before-image table associated with the target temp-table contains a row for the target record, the row is left in place.

Copying ProDataSet object temp-tables in replace mode is less efficient than copying them in append mode. When you are certain the ProDataSet object temp-tables do not contain duplicate rows, copy the ProDataSet object in append mode.

When FALSE, the AVM does not replace records in the target ProDataSet object temp-tables. The default value is FALSE.

loose-copy-mode

An optional logical expression where TRUE indicates that the AVM copy the ProDataSet object temp-tables in a *loose-copy* mode. That is, it relaxes the requirement that the metaschema for the source and target temp-tables be the same.

When TRUE, the AVM copies each temp-table in the source ProDataSet object to the target ProDataSet object based on a field mapping between the source and target temp-table buffers. If there is an attached data source with a field mapping, the AVM uses that field mapping to copy fields from each source temp-table buffer to its target temp-table buffer. If there are fields in either buffer that do not exist in the other, they are ignored. If there is no field mapping with the attached data source, or there is no attached data source, the AVM copies only those fields with the same name that appear in both the source and target temp-table metaschemas.

When FALSE, the metaschema for the source and target temp-tables must be the same or the AVM generates a run-time error. The default value is FALSE.

pairs-list

An optional character expression that evaluates to a comma-delimited list of the target and source temp-table pairs to be copied. Following is the *pairs-list* syntax:

```
"target-table1,source-table1[,target-table2,source-table2]..."
```

If specified, the AVM copies only the listed temp-tables by matching the target and source temp-table names.

If not specified, the AVM copies all the temp-tables in the order they were defined or added in the ProDataSets. If either the source or target ProDataSet has one or more extra temp-tables at the end, the extra temp-tables are ignored.

current-only

An optional logical expression where TRUE indicates that the AVM copy only the current record from each temp-table at each level in the source ProDataSet object to the target ProDataSet object. The default value is FALSE.

Note: You might need to synchronize the buffers to ensure they are the related buffers. Once you have read a record into a top-level buffer, you can synchronize the related buffers by calling the SYNCHRONIZE( ) method.

To copy the current record from a single temp-table, you can use the BUFFER-COPY statement or BUFFER-COPY( ) method.

name-prefix

An optional character expression used as the prefix for naming the target ProDataSet. If the parameter is not passed, or is passed as the Unknown value (?), the AVM uses the default "cpy" prefix. If the parameter is passed as an empty string "", then no prefix is added and the target ProDataSet will have the same name as the source.

When the AVM copies the source ProDataSet object, it copies each temp-table in its entirety and in the order in which the temp-table was defined. That is, it **does not** copy the temp-tables in an interleaved and nested manner based on their data relations. If either the source or target ProDataSet has one or more extra temp-tables at the end, the extra temp-tables are ignored.

When the AVM copies a ProDataSet object in any mode, except loose-copy mode, and the target ProDataSet object has a metaschema (that is, temp-table and relation definitions), the source ProDataSet object metaschema must be the same. If the source and target ProDataSet object metaschema is not the same, the AVM generates a run-time error. If the target ProDataSet object is a newly created dynamic object with only a handle and no metaschema, the AVM copies the source ProDataSet object including its metaschema.

Note

If the source or target (but not both) ProDataSet has any before-table records, you can use COPY-DATASET on that ProDataSet. However, if *append mode* or *replace mode* is specified and the target ProDataSet has any before-table records, the AVM generates a run-time error.

See also

ATTACH-DATA-SOURCE( ) method, FILL-MODE attribute, COPY-TEMP-TABLE( ) method

---

COPY-SAX-ATTRIBUTES( ) method

Performs a deep copy of a SAX-attributes object.

Typically, this method is used to capture the contents of the SAX-attributes object automatically created, populated, and destroyed by the SAX parser during the scope of a call to the startElement callback procedure. For example, to preserve the attributes of an XML element outside of the startElement procedure, you create a SAX-attributes object outside the scope of the startElement callback and use this method on the handle of the new object. The handle to the new SAX-attributes object could then be passed, for example, as a parameter to the START-ELEMENT( ) method of a SAX-writer object. In this way, you can easily program a concurrent SAX read/write routine.

The copy deletes any previous contents of the target handle and replaces them with the contents of the source handle. Remember, if you explicitly create SAX-attributes objects using the CREATE SAX-ATTRIBUTES statement, you must explicitly delete the objects when your copying operations complete.

Return type: LOGICAL

Applies to: SAX-attributes object handle

This is the syntax to invoke this method:

Syntax

```
target-handle:COPY-SAX-ATTRIBUTES ( source-handle )
```

target-handle

A handle to a valid SAX-attributes object. The contents of this SAX-attributes object handle will be overwritten by the contents of the SAX-attributes object of the *source-handle*.

source-handle

A handle to a valid SAX-attributes object. The contents of this SAX-attributes object overwrites the contents of the SAX-attributes object on which you call this method.

The following is an example:

```
hWrite-SAX-attributes:COPY-SAX-ATTRIBUTES( hRead-SAX-attributes ).
```

COPY-TEMP-TABLE( ) method

Copies a source temp-table object to a target temp-table object. Either of the temp-tables (source or target) may be a member of a ProDataSet object. The AVM empties the target temp-table of all records before copying the source temp-table, by default.

Return type: LOGICAL

Applies to: Temp-table object handle, Buffer object handle

Syntax

```
targ-tt-handle:COPY-TEMP-TABLE ( src-tt-handle [ , append-mode
  [ , replace-mode [ , loose-copy-mode [ , name prefix]]]] )
```

targ-tt-handle

The handle to the target temp-table object to receive the copy.

src-tt-handle

The handle to the source temp-table object from which to copy records to the temp-table object on which you call this method.

append-mode

An optional logical expression where TRUE indicates that the AVM copy the temp-table object in an *append* mode.

When TRUE, the AVM appends the source temp-table object to the target temp-table object. If there is a unique index on the target temp-table and the AVM finds a row with a duplicate key, it does not replace the row. If there is not a unique index on the target temp-table, the AVM appends the data row from the source temp-table to the target temp-table, which might result in duplicate rows. If this operation results in duplicate rows, the AVM does not generate a run-time error.

Note: In this mode, the AVM treats duplicate temp-table rows during the copy operation in the same way it treats duplicate temp-table rows during a fill operation in MERGE mode.

When you are certain a temp-table object does not contain duplicate rows, copying the object in append mode is more efficient than copying it in replace mode.

When FALSE, the AVM does not append records in the target temp-table object. The default value is FALSE.

The AVM ignores this expression when *replace-mode* is TRUE.

replace-mode

An optional logical expression where TRUE indicates that the AVM copy the temp-table object in a *replace* mode.

When TRUE, the AVM replaces records in the target temp-table object with corresponding records from the source temp-table object. The target temp-table must have a unique primary index that the AVM can use to find the corresponding record. When the corresponding record is found in the target temp-table, the AVM replaces the target record with the source record. When the corresponding record is not found in the target temp-table, the AVM creates a new target record using the source record. If the before-image table associated with the target temp-table contains a row for the target record, the row is left in place.

Copying a temp-table object in replace mode is less efficient than copying it in append mode. When you are certain a temp-table object does not contain duplicate rows, copy the object in append mode.

When FALSE, the AVM does not replace records in the target temp-table object. The default value is FALSE.

loose-copy-mode

An optional logical expression where TRUE indicates that the AVM copy the temp-table object in a *loose-copy* mode. That is, it relaxes the requirement that the metaschema for the source and target temp-tables be the same.

When TRUE, the AVM copies the source temp-table object to the target temp-table object based on a field mapping between the source and target temp-table buffers. If there is an attached data source with a field mapping, the AVM uses that field mapping to copy fields from the source temp-table buffer to its target temp-table buffer. If there are fields in either buffer that do not exist in the other, they are ignored. If there is no field mapping with the attached data source, or there is no attached data source, the AVM copies only those fields with the same name that appear in both the source and target temp-table metaschema.

When FALSE, the metaschema for the source and target temp-tables must be the same or the AVM generates a run-time error. The default value is FALSE.

name-prefix

An optional character expression used as the prefix for naming the target temp-table. If the parameter is not passed, or is passed as the Unknown value (?), the AVM uses the default "cpy" prefix. If the parameter is passed as an empty string "", then no prefix is added and the target temp-table will have the same name as the source.

If the source temp-table object has a before-image table that contains changed row data, the AVM generates a run-time error.

When the AVM copies a temp-table object in any mode, except loose-copy mode, and the target temp-table object is in a PREPARED state (that is, it has a metaschema), the source temp-table object metaschema must be the same. Each column in the source temp-table must match the target temp-table in position, data type, and extent. If the source temp-table object metaschema is not the same, the AVM generates a run-time error. If the target temp-table object is not in a PREPARED state (that is, it has no metaschema), the AVM copies the source temp-table object including its metaschema.

If the target temp-table object is a member of a ProDataSet object, the AVM does not track changes to the data in that temp-table (it ignores the TRACKING-CHANGES attribute setting during the copy operation).

Note

If the source or target (but not both) temp-table has any before-table records, you can use COPY-TEMP-TABLE on that temp-table. However, if *append mode* or *replace mode* is specified and the target temp-table has any before-table records, the AVM generates a run-time error.

See also

FILL-MODE attribute, COPY-DATASET( ) method

COVERAGE attribute

Determines whether or not the profiler records information that can be used for coverage analysis. The default is FALSE.

Data type: LOGICAL

Access: Readable/Writable

Applies to: PROFILER system handle

If set to TRUE, the profiler records statement and internal procedure information for each external `.p` procedure that is executed. If set to FALSE, then this level of information is not collected. Combined with the timing data that tells which statements were executed, can be used to perform ABL application code coverage analysis. For example, collecting coverage information could show that during testing only 30 percent of the code was executed.

Note: The profiler records the information for coverage analysis only the first time an external `.p` procedure is executed. If the PROFILER:COVERAGE attribute is FALSE, then the first time a procedure is executed the coverage analysis information is not registered, even if it is set to TRUE at a later time.

CPCASE attribute

The case table the AVM uses to establish case rules for the Internal Code Page (`-cpinternal`) startup parameter.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

This attribute reads the value you set using the Case Table (`-cpcase`) startup parameter.

CPCOLL attribute

The collation table the AVM uses with the Internal Code Page (`-cpinternal`) startup parameter.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

By default, the AVM uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (`-cpcoll`) startup parameter take precedence over a collation specified for any database the AVM accesses during the session, except when the AVM uses or modifies pre-existing indexes. If you do not specify a collation with the `-cpcoll` startup parameter, the AVM uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, the AVM uses the collation rules with the default name "`basic`" (which might or might not exist in the `convmap.cp` file).

CPINTERNAL attribute

The internal code page the AVM uses in memory.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

This attribute reads the value you set using the Internal Code Page (`-cpinternal`) startup parameter.

CPLOG attribute

The code page for all messages written to the log (`.lg`) file.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

This attribute reads the value you set using the Log File Code Page (`-cplog`) startup parameter.

CPPRINT attribute

The code page the AVM uses for the OUTPUT TO PRINTER statement.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

This attribute reads the value you set using the Printer Code Page (`-cpprint`) startup parameter.

CPRCODEIN attribute

The code page the AVM uses to convert text strings into the text segment.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

This attribute reads the value you set using the R-code In Code Page (`-cprcodein`) startup parameter.

CPRCODEOUT attribute

The code page ABL uses at compile time to convert text strings into the text segment and marks the text segment with the code page name.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

This attribute reads the value you set using the R-code Out Code Page (`-cprcodeout`) startup parameter.

CPSTREAM attribute

The code page the AVM uses for stream I/O.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

This attribute reads the value you set using the Stream Code Page (`-cpstream`) startup parameter.

CPTERM attribute

The code page the AVM uses for I/O with character terminals.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

This attribute reads the value you set using the Terminal Code Page (`-cpterm`) startup parameter.

CRC-VALUE attribute

The cyclic redundancy check (CRC) value for either an r-code file, or a database table corresponding to a buffer object.

Data type: INTEGER

Access: Read-only

Applies to: Buffer object handle, RCODE-INFO handle

When applied to the RCODE-INFO system handle, the r-code CRC is calculated during compilation and stored in the r-code file using the contents of the source files, including included files.

When applied to the Buffer object handle, the database CRC is calculated using the metaschema _CRC field value from the _File record for the database record corresponding to the buffer object's table (which can be a standard or temp-table).

The CRC for a temp-table is calculated differently than for a standard table. Some differences include:

- Standard tables have a _File record in the database that contains a _CRC field for the CRC value, which is calculated as you make changes to the table. Temp-tables do not have a _File record, and do not exist in a database.
- The CRC values for both standard and temp-tables include the data type, extent and position of each column in the table, as well as index information. However, the CRC value for a standard table includes additional information that a CRC value for a temp-table does not (such as, the _Order field in the _File record).
- The CRC value for a standard table is stored in a .r file. The AVM uses that CRC value at run time to verify the integrity of application r-code that uses the table. The AVM uses the CRC value for a temp-table to compare table parameters between a calling and called procedure (to avoid a field-by-field comparison).

For more information on CRCs, see *Manage ABL Applications*.

CREATE-LIKE( ) method

Creates a table like another existing table, or a dynamic ProDataSet object like another static or dynamic ProDataSet object.

Return type: LOGICAL

Applies to: ProDataSet object handle, Temp-table object handle

Syntax

```
CREATE-LIKE ( { src-buffer-handle-exp | src-table-handle-exp |
src-table-name-exp }
  [ , src-index-name-exp ] )
```

```
CREATE-LIKE ( { src-dataset-handle | src-dataset-name }
  [ , name-prefix ] )
```

src-buffer-handle-exp

An expression that evaluates to a buffer handle from which to copy the field definitions, and optionally, the indexes if *src-index-name-exp* is not specified. This can be a buffer for either a temp-table or database.

src-table-handle-exp

An expression that evaluates to a temp-table handle from which to copy the field definitions, and optionally, the indexes if *src-index-name-exp* is not specified.

src-table-name-exp

An expression that evaluates to the name of a temp-table or database table from which to copy the field definitions and, optionally, the indexes if *src-index-name-exp* is not specified.

src-index-name-exp

A character expression giving an index to be copied from the source table. If this option is specified, only this single index is copied from the source table.

src-dataset-handle

The handle to the ProDataSet object from which to create the new ProDataSet object.

src-dataset-name

The name of the ProDataSet object from which to create the new ProDataSet object.

name-prefix

A character expression to prepend to each of the source ProDataSet member buffer names, which creates a new name for each new member buffer.

For a table handle, this method copies the field definitions from the specified source table and establishes the default or specified source indexes. You cannot call this method after another definitional method is called unless you call CLEAR( ) first.

For a ProDataSet object handle, this method creates a dynamic ProDataSet object like another static or dynamic ProDataSet object. The AVM creates the new ProDataSet object with the same name, temp-table definitions, and relation definitions. The AVM also creates the same before-image and after-image tables, if any exist for the source object. No data from the source temp-tables is copied. The AVM also lets you rename the newly created ProDataSet member buffers by prepending a prefix to the source buffer names.

Note: For a ProDataSet object handle, if you call CLEAR( ) or CREATE-LIKE( ) more than once on a dataset, the dynamic temp-tables from the prior call(s) are not deleted automatically and have the potential to cause a memory leak. Therefore the application is responsible for deleting the temp-tables if CLEAR( ) or CREATE-LIKE( ) are called again.

See also

CREATE-LIKE-SEQUENTIAL( ) method

CREATE-LIKE-SEQUENTIAL( ) method

Creates a table like another existing database table or temp-table. Unlike CREATE-LIKE( ), which creates temp-table fields in metaschema _field._field-rpos order (POSITION order in the .df schema definition file) of the source table's fields, this method creates fields in _field._order sequence.

Return type: LOGICAL

Applies to: Temp-table object handle

You can guarantee agreement of temp-table field order between any client and any application server using CREATE-LIKE-SEQUENTIAL( ), as long as the _field._order values are the same. CREATE-LIKE-SEQUENTIAL( ) uses the field order as defined in the Data Dictionary when the source is a database table.

Syntax

```
CREATE-LIKE-SEQUENTIAL ( { src-buffer-handle-exp | src-table-handle-exp |
src-table-name-exp }
  [ , src-index-name-exp ] )
```

src-buffer-handle-exp

An expression that evaluates to a buffer handle from which to copy the field definitions, and optionally, the indexes if *src-index-name-exp* is not specified. This can be a buffer for either a temp-table or database.

src-table-handle-exp

An expression that evaluates to a temp-table handle from which to copy the field definitions, and optionally, the indexes if *src-index-name-exp* is not specified.

src-table-name-exp

An expression that evaluates to the name of a temp-table or database table from which to copy the field definitions and, optionally, the indexes if *src-index-name-exp* is not specified.

src-index-name-exp

A character expression giving an index to be copied from the source table. If this option is specified, only this single index is copied from the source table.

This method copies the field definitions from the specified source table and establishes the default or specified source indexes. You cannot call this method after another definitional method is called unless you call CLEAR( ) first.

Note: The original behavior of CREATE-LIKE( ) was used to support RAW-TRANSFER with temp-tables. If you are using RAW-TRANSFER between a database table and a temp-table defined like the database table, then you should not use CREATE-LIKE-SEQUENTIAL( ).

See also

CREATE-LIKE( ) method

CREATE-NODE( ) method

Create an XML node in the current document. The first parameter must be a valid X-noderef handle and will refer to the new XML node if the method succeeds. This method merely creates the XML node as part of the XML document. The INSERT-BEFORE or APPEND-CHILD methods are required to actually insert it into the document's tree.

Return type: LOGICAL

Applies to: X-document object handle

Syntax

```
CREATE-NODE ( x-node-handle , name , type )
```

x-node-handle

A valid X-noderef handle to use for the new XML node.

name

A character expression representing the NAME of the node. The relationship between the node NAME and SUBTYPE attributes is shown in the following table.

type

A character expression representing the node's SUBTYPE, which will be one of: ATTRIBUTE, CDATA-SECTION, COMMENT, DOCUMENT-FRAGMENT, ELEMENT, ENTITY-REFERENCE, PROCESSING-INSTRUCTION, TEXT.

Table 74: Relationship between the SUBTYPE and NAME attributes

If the SUBTYPE is . . .	then the NAME attribute is . . .
ATTRIBUTE	The name of the attribute
CDATA-SECTION COMMENT DOCUMENT-FRAGMENT TEXT	A constant value; the *name* parameter is ignored
ELEMENT	The name of the XML tag, with any namespace prefix included
ENTITY-REFERENCE	The name of the entity referenced without leading ampersand and trailing semicolon
PROCESSING-INSTRUCTION	The target; the first token following the <? markup

The following example demonstrates creating a node in a document. If hDoc is an X-document handle, and hNoderef and hNoderefChild are X-noderefs, this is how you would add hNoderefChild to hNoderef in the document associated with hDoc:

```
/* Assume hNoderef has previously been added to the tree. Create an ABL handle
   that can refer to a node in an XML parse tree.*/
CREATE X-NODEREF hNoderefChild.

/* Create an XML node whose name is "Address" & whose type is "ELEMENT" */
hDoc:CREATE-NODE(hNoderefChild, "Address", "ELEMENT")

/* Put this child into the tree and ultimately into the document. */
hNoderef:APPEND-CHILD(hNoderefChild).
```

CREATE-NODE-NAMESPACE( ) method

Creates a namespace-aware XML node whose name can be either a single string `y` or an `x:y` combination.

Note: To ensure consistency across all nodes in an XML document, use either the CREATE-NODE-NAMESPACE( ) method or the CREATE-NODE( ) method to build an XML document; do not use both methods within a single document.

Return type: LOGICAL

Applies to: X-document object handle

Syntax

```
CREATE-NODE-NAMESPACE
  ( x-node-handle , namespace-uri , qualified-name , type )
```

x-node-handle

A valid X-noderef handle to use for the new namespace-aware XML node.

namespace-uri

A character expression representing the namespace Uniform Resource Identifier (URI). The *namespace-uri* must be unique and persistent. Although the *namespace-uri* may be an HTTP URL, there is no guarantee that it points to a retrievable resource. It is only a name and care should be taken if you use this name for other purposes.

If the character expression evaluates to either the empty string ("") or the Unknown value (?), no namespace is associated with the element.

qualified-name

A character expression representing the name of the node, optionally qualified with a prefix including a colon (for example, *prefix:node-name*). Unless you are using a default namespace, a prefix is required and should be set to the prefix specified when you declared the namespace using the xmlns attribute.

type

A character expression representing the node's SUBTYPE, which will be either ELEMENT or ATTRIBUTE.

The following code fragment illustrates how to create a namespace-aware node in an XML document using either a specific namespace or the default namespace:

```
. . .
/* Look for a colonized name in rootNodeName. */
found = INDEX(rootNodeName, ":").
IF found > 0 THEN DO:
  /* Namespace declarations are special kinds of attributes that belong in
     the http://www.w3.org/2000/xmlns/ namespace.*/

 errStat = hDocument:CREATE-NODE-NAMESPACE(hNsDecl,
  "http://www.w3.org/2000/xmlns/", "xmlns:" +
  SUBSTRING(rootNodeName, 1, found - 1), "attribute").
END.

ELSE DO:
 /* Use the default namespace, which does not need a namespace declaration
     prefix, and assign it to the http://www.w3.org/2000/xmlns/ namespace.*/

 errStat = hDocument:CREATE-NODE-NAMESPACE(hNsDecl,
  "http://www.w3.org/2000/xmlns/", "xmlns", "attribute").
END.

IF NOT errStat THEN LEAVE.

/* Set the value of the namespace attribute to the namespace URI. */
hNsDecl:NODE-VALUE = namespaceURI.
. . .
```

CREATE-RESULT-LIST-ENTRY( ) method

Creates an entry in the result list for the current row. The developer uses the CREATE-RESULT-LIST-ENTRY method in conjunction with new browse rows or new query rows to synchronize the data with the query.

Return type: LOGICAL

Applies to: BROWSE widget, Query object handle

Syntax

```
CREATE-RESULT-LIST-ENTRY ( )
```

For browses, this method should be used with a ROW-LEAVE trigger when the NEW-ROW attribute is TRUE. For example, if the user adds a row to an updateable browse, you can use this method to create an entry for the new row in the result list.

CURRENT-CHANGED attribute

Indicates whether a record in a buffer is different following a FIND CURRENT or GET CURRENT statement or method. If the record is different, CURRENT-CHANGED is TRUE. Otherwise, CURRENT-CHANGED is FALSE.

Data type: LOGICAL

Access: Read-only

Applies to: Buffer object handle

Note: The CURRENT-CHANGED attribute corresponds to the CURRENT-CHANGED function.

CURRENT-COLUMN attribute

The value of the browse column that contains the current cell. This attribute moves focus to the cell in the specified column in the current row.

Data type: HANDLE

Access: Readable/Writeable

Applies to: BROWSE widget

For the browse, if the browse or a browse component currently has focus, then setting the attribute to another column causes the proper LEAVE and cell ENTRY events to happen.

If the setting of the CURRENT-COLUMN attribute happens when focus is outside of the browse, then the browse's internal handle to the current column is updated so that it will become the current column when you tab back into the browse. Also if you apply "START-SEARCH" the search mode will now use this column to search on.

CURRENT-ENVIRONMENT attribute

Returns a list of CGI environment variable settings and HTTP header information, and is used by the `get-cgi` WebSpeed API function. Intended for internal use only.

Data type: CHARACTER

Access: Read-only

Applies to: WEB-CONTEXT system handle

CURRENT-ITERATION attribute (Data Objects)

Indicates which iteration level corresponds to the buffer handle during a recursive `FILL` of a ProDataSet.

Data type: INTEGER

Access: Read-only

Applies to: Buffer object handle

As a recursive ProDataSet `FILL` is proceeding, it creates clones of the relevant buffers, relations, queries and data-sources for each level of recursion. As a new record is added to the ProDataSet, it fires `FILL` events on the recursed, cloned buffer. From inside the event handler, you may want to see previous iterations of the buffer; its parent, grandparent, great-grandparent, and so on. `CURRENT-ITERATION` returns the level of iteration for the cloned buffer handle. To refer to the cloned buffer handle, use `SELF:CURRENT-ITERATION` from within the `BEFORE-ROW-FILL` and `AFTER-ROW-FILL` event handlers.

If the buffer object handle does not point to a ProDataSet temp-table or is not part of a recursive data-relation, `CURRENT-ITERATION` returns the Unknown value (?).

See also

DEFINE DATASET statement, FILL( ) method, GET-ITERATION( ) method (Data Objects), NUM-ITERATIONS attribute (data objects), RECURSIVE attribute

CURRENT-ITERATION attribute (Widget Objects)

A handle for the current iteration of the frame or dialog box.

Data type: HANDLE

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget

This attribute is a read-only attribute for dialog boxes.

CURRENT-QUERY( ) method

The handle to the ProDataSet query that contains the currently selected row in the ProBindingSource. This method enables access to the correct child query when a bound .NET grid edits, creates, or deletes a row from a child table.

Use this method when the .NET control is bound to a ProDataSet object that contains child buffers.

Data type: HANDLE

Access: Read-only

Applies to: Data-relation object handle

Syntax

```
CURRENT-QUERY ( [band-index] )
```

band-index

An optional integer value that specifies the band index in a hierarchical grid. The band index is a 0-based index. Use this parameter only with recursive data-relations. If you specify this parameter for a non-recursive data-relation, the AVM raises an error.

When a .NET control binds to a ProBindingSource that uses a ProDataSet as its data source object, multiple relations for the same parent-child relationship might be presented to the user simultaneously. When the user first expands a parent row in the bound .NET control, the ProBindingSource creates a new child query and displays all child rows for that parent row. The ProBindingSource uses the ProDataSet object's data-relation query to create and maintain this unique query for all child rows related to the expanded parent row.

Note: You cannot set this query handle or modify the query.

The following code fragment shows an event handler. The event handler accepts information from a bound UltraWinGrid displaying Order and Orderline data through the grid's extended version of the .NET `System.EventArgs` class. From that information, the event handler can find the correct query for the currently selected record.

```
DEFINE VARIABLE hDataSet AS HANDLE NO-UNDO.

...

PROCEDURE BeforeRowUpdate:
  DEFINE INPUT PARAMETER sender AS System.Object         NO-UNDO.
  DEFINE INPUT PARAMETER args   AS
    Infragistics.Win.UltraWinGrid.CancelableRowEventArgs NO-UNDO.

  DEFINE VARIABLE cBufferName AS CHARACTER NO-UNDO.
  DEFINE VARIABLE hRelation   AS HANDLE    NO-UNDO.
  DEFINE VARIABLE hQuery      AS HANDLE    NO-UNDO.

  cBufferName = args:Row:Band:Key.
  IF cBufferName EQ "ttOrder" THEN
    ASSIGN
      hRelation = hDataSet:GET-RELATION(1)
      hQuery    = hRelation:CURRENT-QUERY.
  END.
  ELSE IF cBufferName EQ "ttOrderLine" THEN
    ASSIGN
      hRelation = hDataSet:GET-RELATION(2)
      hQuery    = hRelation:CURRENT-QUERY.
  END.
END PROCEDURE.
```

When a ProBindingSource is attached to a ProDataSet with a recursive data-relation, the code above does not work because a single buffer is associated with multiple BandIndexes in a hierarchical grid. The event handler must specify the BandIndex of the current row to determine the correct query.

The following example shows a similar event handler using the optional parameter. Because an employee might also be a manager and have employees of his own, dsRecursive has a recursive data-relation between EmpName and Manager. For the top band of the hierarchy, the event handler can use the TOP-NAV-QUERY. For lower bands, it needs the optional parameter to determine the correct query.

```
DEFINE TEMP-TABLE ttEmp NO-UNDO
  BEFORE-TABLE ttEmpB
  FIELD EmpName    AS CHARACTER
  FIELD Manager    AS CHARACTER
  FIELD HireDate   AS INTEGER
  INDEX idxEmpName AS UNIQUE EmpName.

DEFINE DATASET dsRecursive FOR ttEmp
  DATA-RELATION r1 FOR ttemp, ttemp
  RELATION-FIELDS(EmpName,Manager) RECURSIVE.

DEFINE VARIABLE hRelation AS HANDLE NO-UNDO.
DEFINE VARIABLE hTopQuery AS HANDLE NO-UNDO.

...

PROCEDURE BeforeRowUpdate:
  DEFINE INPUT PARAMETER sender AS System.Object         NO-UNDO.
  DEFINE INPUT PARAMETER args   AS
    Infragistics.Win.UltraWinGrid.CancelableRowEventArgs NO-UNDO.
  DEFINE VARIABLE bandIndex     AS INTEGER               NO-UNDO.
  DEFINE VARIABLE hQuery        AS HANDLE                NO-UNDO.

  ASSIGN
    bandIndex = args:Row:Band:Index
    hRelation = DATASET dsRecursive:GET-RELATION(1).

  IF bandIndex EQ 0 THEN
    hQuery = hTopQuery.
  ELSE IF bandIndex > 0 THEN
    hQuery = hRelation:CURRENT-QUERY(bandIndex).
END PROCEDURE.
```

See also

BandIndex property

CURRENT-REQUEST-INFO attribute

An object reference to a `Progress.Lang.OERequestInfo` class that provides information about the current client request sent to and executing on an application server session.

Note: When using the Progress Application Server (PAS) for OpenEdge, this attribute is valid on both the client and server sides.

Data type: Progress.Lang.OERequestInfo class

Access: Read-only

Applies to: SESSION system handle

The following application server code fragment retrieves the client context identifier (CCID) that has been sent with a current request from an application server client:

```
DEFINE VARIABLE ccid AS CHARACTER NO-UNDO.
ASSIGN
  ccid = SESSION:CURRENT-REQUEST-INFO:ClientContextId.
```

The bold code in the following fragment assigns a copy of the client principal in the current session:

```
DEFINE VARIABLE hCPIn   AS HANDLE.
DEFINE VARIABLE hCPOut  AS HANDLE.

hCPIn = SESSION:CURRENT-REQUEST-INFO:GetClientPrincipal().
SECURITY-POLICY:SET-CLIENT(hCPIn).
hCPOut = SECURITY-POLICY:GET-CLIENT().
SESSION:CURRENT-RESPONSE-INFO:SetClientPrincipal(hCPOut).
```

See also

GetClientPrincipal( ) method (Progress.Lang.OERequestInfo), CURRENT-RESPONSE-INFO attribute, LOCAL-VERSION-INFO attribute, REQUEST-INFO attribute, RESPONSE-INFO attribute, SetClientPrincipal( ) method

CURRENT-RESPONSE-INFO attribute

An object reference to a `Progress.Lang.OERequestInfo` class that allows you to set the response to a current request that an application server session returns to a client.

Note: When using the Progress Application Server (PAS) for OpenEdge, this attribute is valid on both the client and server sides.

Data type: Progress.Lang.OERequestInfo class

Access: Readable/Writeable

Applies to: SESSION system handle

The following application server code fragment generates a unique value as a client context identifier (CCID) and assigns it as the value returned to an application server client in response to the current request:

```
DEFINE VARIABLE ccid AS CHARACTER NO-UNDO.
ccid = SUBSTRING(BASE64-ENCODE (GENERATE-UUID), 1, 22).
ASSIGN
  SESSION:CURRENT-RESPONSE-INFO:ClientContextId = ccid.
```

The bold code in the following fragment sets the client principal returned to an application server client:

```
DEFINE VARIABLE hCPIn   AS HANDLE.
DEFINE VARIABLE hCPOut  AS HANDLE.

hCPIn = SESSION:CURRENT-REQUEST-INFO:GetClientPrincipal().
SECURITY-POLICY:SET-CLIENT(hCPIn).
hCPOut = SECURITY-POLICY:GET-CLIENT().
SESSION:CURRENT-RESPONSE-INFO:SetClientPrincipal(hCPOut).
```

See also

GetClientPrincipal( ) method (Progress.Lang.OERequestInfo), CURRENT-REQUEST-INFO attribute, LOCAL-VERSION-INFO attribute, REQUEST-INFO attribute, RESPONSE-INFO attribute, SetClientPrincipal( ) method

CURRENT-RESULT-ROW attribute

The sequence number of the current row of a dynamic query's result list.

Data type: INTEGER

Access: Read-only

Applies to: Query object handle

Note: The CURRENT-RESULT-ROW attribute corresponds to the CURRENT-RESULT-ROW function.

See also

CURRENT-RESULT-ROW function

CURRENT-ROW-MODIFIED attribute

Indicates whether any cells in the current row have been changed.

Data type: LOGICAL

Access: Read-only

Applies to: BROWSE widget

The CURRENT-ROW-MODIFIED attribute is set to TRUE if the user has modified any cell within the current row since focus moved to that row.

CURRENT-WINDOW attribute

A current window for the specified procedure.

Data type: HANDLE

Access: Readable/Writeable

Applies to: Procedure object handle, SOURCE-PROCEDURE system handle, TARGET-PROCEDURE system handle, THIS-PROCEDURE system handle

Specifies and allows you to reset the current window used to parent alert box, dialog box, or frame widgets for the specified procedure. The default value is the Unknown value (?). Returns the Unknown value (?) for a Web service procedure or proxy procedure.

If you set this attribute to the handle of a window, this value takes precedence over the CURRENT-WINDOW handle to provide the default window for parenting alert boxes, frames, and dialog boxes created within the procedure.

This attribute is especially useful for creating and associating a unique current window with each instantiation of a persistent procedure. For more information on persistent procedures, see the RUN statement reference entry.

CURSOR-CHAR attribute

The current character position of the text cursor on the current text line in an editor widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: EDITOR widget

Assigning a value to CURSOR-CHAR moves the text cursor to the specified character position on the current text line. If the editor widget is not already realized, the AVM realizes the widget when you query the CURSOR-CHAR attribute.

CURSOR-LINE attribute

The line within an editor widget where the text cursor is positioned.

Data type: INTEGER

Access: Readable/Writeable

Applies to: EDITOR widget

Assigning a value to CURSOR-LINE moves the text cursor to the specified line. If the editor widget is not already realized, the AVM realizes the widget when you query the CURSOR-LINE attribute.

CURSOR-OFFSET attribute

The character offset of the cursor within a widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (cell), EDITOR widget, FILL-IN widget

Assigning a value to CURSOR-OFFSET moves the text cursor to the specified character offset. If the editor widget is not already realized, the AVM realizes the widget when you query the CURSOR-OFFSET attribute.

In Windows, both the regular editor and the large editor support CURSOR-OFFSET.

For browses, this attribute applies only to fill-in and combo-box browse columns.

For browse columns, CURSOR-OFFSET specifies the character offset of the cursor within a browse-cell of the browse column.

DATA-ENTRY-RETURN attribute

The behavior of the **RETURN** key for the fill-in widgets of a frame.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

If TRUE, the **RETURN** key in a fill-in acts like a **TAB**, and if the fill-in is the last widget in the tab order of its parent frame and of all ancestor frames, the **RETURN** key applies a GO event to the frame. This GO event, from a fill-in **RETURN**, propagates to all ancestor frames and their descendants, including siblings of the current frame and their descendants, all in the same frame family. If a widget is not a fill-in, the window system handles **RETURN** entries.

The default value is TRUE for character interfaces and FALSE for graphical interfaces. The AVM ignores this attribute if there is a default button on the frame.

DATA-SOURCE attribute

Returns the handle to the data-source object currently attached to the ProDataSet object buffer.

Data type: HANDLE

Access: Read-only

Applies to: Buffer object handle

DATA-SOURCE-COMPLETE-MAP attribute

Returns a comma-delimited list of field name pairs for all fields in a ProDataSet temp-table buffer that are mapped to corresponding fields in an attached data-source object.

Data type: CHARACTER

Access: Read-only

Applies to: Buffer object handle

This list is formatted as a comma-delimited list of field name pairs, qualified with the corresponding ProDataSet temp-table and data-source object names, using the following syntax:

Syntax

```
tt-buffer-name.tt-field-name,db-table-name.db-field-name
[ , tt-buffer-name.tt-field-name,db-table-name.db-field-name ] ...
```

Note: You may use a subscript reference for array fields mapped explicitly through subscripts.

If the ProDataSet temp-table buffer does not have an attached data-source object, this attributes returns the Unknown value (?).

Use the ATTACHED-PAIRLIST attribute to get a list of only the field name pairs you specified with the most recently attached data-source object.

DATA-SOURCE-MODIFIED attribute

Indicates that data in the data source associated with a ProDataSet temp-table buffer has been modified.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

The DATA-SOURCE-MODIFIED attribute corresponds to the DATA-SOURCE-MODIFIED function.

The AVM sets this attribute from the SAVE-ROW-CHANGES( ) method. You can also set this attribute, if needed.

This attribute is marshalled between the client and the application server.

DATA-SOURCE-ROWID attribute

Identifies the data-source row corresponding to the buffer object handle.

Data type: ROWID

Access: Readable/Writeable

Applies to: Buffer object handle

This attribute locates the data-source row corresponding to either an after-table buffer or a before-table buffer. The portion of the SAVE-ROW-CHANGES( ) method that locates the appropriate data-source row is used to produce a ROWID for users who want to either make their own version of SAVE-ROW-CHANGES( ) or access the source record in some way before or after SAVE-ROW-CHANGES( ).

Syntax

```
DATA-SOURCE-ROWID ( { join-level | buffer-name } )
```

join-level

An integer expression that evaluates to the level of join, if the data-source is a join.

buffer-name

A character expression that evaluates to the name of one of the data-source buffers.

See also

AFTER-TABLE attribute, BEFORE-TABLE attribute, SAVE-ROW-CHANGES( ) method

DATA-TYPE attribute

A character value that represents the data type of the field associated with the widget. For example, the DATA-TYPE attribute of a slider widget always returns the value "INTEGER" because slider widgets can only represent integer values.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget (column), Buffer-field object handle, COMBO-BOX widget, EDITOR widget, FILL-IN widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

This attribute is writeable for combo-boxes (only before realization), fill-ins, and text widgets only. For combo-boxes, writing to this attribute makes the drop-down list empty.

You must define this attribute as "CHARACTER" for SIMPLE and DROP-DOWN combo-boxes.

For widgets like image or rectangle, where a data type has no meaning, the attribute returns "UNKNOWN".

The DATA-TYPE attribute is only writable for dynamic fill-ins before they are realized. This attribute is read only for static fill-ins.

DATASET attribute

Returns the handle for the ProDataSet object of which the buffer is a member. Use this handle to access the attributes and methods of the associated ProDataSet object.

Data type: HANDLE

Access: Read-only

Applies to: Buffer object handle

DATE-FORMAT attribute

The format used to represent dates during the current ABL session (for the DATE, DATETIME, and DATETIME-TZ data types).

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SESSION system handle

Typical values are "mdy" or "dmy". This attribute provides the same functionality as the Date Format (-d) parameter.

DB-CONTEXT attribute

Returns the logical database to which the DSLOG-MANAGER system handle currently applies. Valid values are any valid, connected logical name or alias for a database of a supported database type. The default value is the Unknown value (?). If you set DB-CONTEXT to an alias, it returns the logical name of the database currently assigned to that alias.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: DSLOG-MANAGER system handle

Each DataServer connection is a separate entity for which you can specify different logging settings. You use this attribute at run time to specify the DataServer connection with which you are working. You must set DB-CONTEXT before accessing any other attribute or method of the DSLOG-MANAGER system handle.

If you set DB-CONTEXT to the Unknown value (?), you cannot access DSLOG-MANAGER until you reset DB-CONTEXT to a valid logical database name or alias. When the database specified in DB-CONTEXT is disconnected, the client automatically sets DB-CONTEXT to the Unknown value (?).

The following statements show how you can set this attribute:

```
DSLOG-MANAGER:DB-CONTEXT = "mymss".

DSLOG-MANAGER:DB-CONTEXT = myvar.

DSLOG-MANAGER:DB-CONTEXT = "myalias".
```

DB-LIST attribute

A comma-separated list of logical database names that is stored in the client-principal object for all OpenEdge multi-tenant databases for which a connection identity has been set in one or more ABL sessions using this object. A given logical database name is stored in the client-principal object whenever a user authentication or single-sign-on operation using this object sets the multi-tenant database connection identity. You can use a database name from this list as input to the TENANT-ID( ) and TENANT-NAME( ) methods in order to retrieve the corresponding tenant name and tenant ID that has been stored along with it.

Note: The list returned by this function **omits** the names of all **non-multi-tenant** databases for which the connection identity has been set using this client-principal object.

Data type: CHARACTER

Access: Read-only

Applies to: Client-principal object handle

This attribute returns the:

- **Unknown value (?)** — If the client-principal object is not sealed
- **Blank ("") value** — If the client-principal object is sealed but has not yet been used to set identity for any OpenEdge database connections

Tenant information is stored in the client-principal for each database connection whose identity is established by a user authentication or single sign-on (SSO) operation using this object (see the CONNECT statement, SET-DB-CLIENT function, SET-CLIENT( ) method, and SETUSERID function entries). Once stored in the client-principal, the tenant information for a given database never changes, regardless of how many ABL sessions use the object to set connection identity and regardless of the status of database connections in the current ABL session. Therefore, the existing tenant information is carried with the object as it is used to set additional database connection identities and as it is exported (using the EXPORT-PRINCIPAL( ) method) from one ABL session to another. This information can be removed only if the client-principal object is deleted or is re-initialized (using the INITIALIZE( ) method).

ABL raises ERROR if you attempt to assign a value to this attribute.

See also

TENANT-ID( ) method, TENANT-NAME( ) method

DB-REFERENCES attribute

A comma-separated list of the databases, (in the form of logical database names) referenced by an r-code file or by a persistent procedure. Returns the Unknown value (?) for a Web service procedure or proxy procedure.

Data type: CHARACTER

Access: Read-only

Applies to: Procedure object handle, RCODE-INFO handle, SOURCE-PROCEDURE system handle, TARGET-PROCEDURE system handle, THIS-PROCEDURE system handle

The following example displays a list of all databases referenced by sample.r in a comma-separated list that is contained in the DB-REFERENCES attribute.

```
RCODE-INFO:FILE-NAME = "sample.r".
DISPLAY RCODE-INFO:DB-REFERENCES.
```

DBNAME attribute

The logical name of the database from which the field is taken.

Data type: CHARACTER

Access: Read-only

Applies to: BROWSE widget (column), Buffer object handle, Buffer-field object handle, COMBO-BOX widget, EDITOR widget, FILL-IN widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

The default logical name for a database name is the name of the database file without the file extension. You can set the logical name of a database using the Logical Database Name (-ld) parameter.

DCOLOR attribute
(Character Interfaces only)

The color number for the display color of the widget in character mode. This attribute is ignored in graphical interfaces.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse, column, and cell), BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, LITERAL widget, MENU widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

For browse widgets, it is readable only. For a browse cell, it specifies the color of a specific cell in the view port. You can set this color only as the cell appears in the view port during a ROW-DISPLAY event.

For rectangles, DCOLOR specifies the fill color. In windows, DCOLOR specifies the color inherited by the menu bar, if the menu bar has no color specified.

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

You can now change the color of the background of menu frames (including menubars, submenus and pop-up menus) using the DCOLOR attribute. Previously, specifying the DCOLOR attribute for menus only changed the default color for menu items. Now, the DCOLOR attribute will be applied to the menu frame also. Note that no syntax changes were made. You can still specify the DCOLOR attribute for individual menu items.

For more information on widget color, see the PFCOLOR attribute.

DDE-ERROR attribute
(Windows only)

The error condition returned by the most recent exchange in a DDE conversation associated with the frame. A DDE function or a DDE-NOTIFY event initiates an exchange in a DDE conversation.

This attribute is supported only for backward compatibility. Use the Component Object Model (COM) instead.

Data type: INTEGER

Access: Read-only

Applies to: FRAME widget

The following table lists the possible errors returned by an exchange in a DDE conversation.

Table 75: ABL DDE errors

Error	Description
1	DDE INITIATE failure
2	A DDE statement (DDE ADVISE, DDE EXECUTE, DDE GET, DDE REQUEST, or DDE SEND) time out
3	Memory allocation error
4	Invalid channel number (not an open conversation)
5	Invalid data item (in topic)
6	DDE ADVISE failure (data link not accepted)

Error	Description
7	DDE EXECUTE failure (commands not accepted)
8	DDE GET failure (data not available)
9	DDE SEND failure (data not accepted)
10	DDE REQUEST failure (data not available)
11	DDE-NOTIFY event failure (data not available)
99	Internal error (unknown)

This attribute applies to any frame in Windows that is a Dynamic Data Exchange (DDE) frame for a DDE conversation.

DDE-ID attribute
(Windows only)

The DDE channel number of the most recent conversation involved in an exchange.

This attribute is supported only for backward compatibility. Use the Component Object Model (COM) instead.

Data type: INTEGER

Access: Read-only

Applies to: FRAME widget

This attribute applies to any frame in Windows that is a Dynamic Data Exchange (DDE) frame for a DDE conversation.

DDE-ITEM attribute
(Windows only)

The name of the data item affected by the most recent conversational exchange (for example, the name of a worksheet cell such as "R3C5").

This attribute is supported only for backward compatibility. Use the Component Object Model (COM) instead.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: FRAME widget

This attribute applies to any frame in Windows that is a Dynamic Data Exchange (DDE) frame for a DDE conversation.

DDE-NAME attribute
(Windows only)

The name of the application involved in the most recent conversational exchange (for example, the name of a worksheet application such as "EXCEL").

This attribute is supported only for backward compatibility. Use the Component Object Model (COM) instead.

Data type: CHARACTER

Access: Read-only

Applies to: FRAME widget

This attribute applies to any frame in Windows that is a Dynamic Data Exchange (DDE) frame for a DDE conversation.

DDE-TOPIC attribute
(Windows only)

The topic name of the most recent conversation (for example, the "System" topic, or the name of an Excel worksheet such as "Sheet1").

This attribute is supported only for backward compatibility. Use the Component Object Model (COM) instead.

Data type: CHARACTER

Access: Read-only

Applies to: FRAME widget

This attribute applies to any frame in Windows that is a Dynamic Data Exchange (DDE) frame for a DDE conversation.

DEBLANK attribute

How to process leading blanks in fill-in widgets during user input.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FILL-IN widget

This attribute applies to fill-ins for CHARACTER fields that are enabled for input.

If the DEBLANK attribute is TRUE, the AVM removes leading blanks from the widget following user input that changes the SCREEN-VALUE attribute of the widget. Any leading blanks in the SCREEN-VALUE before input are not removed unless the user modifies the field. After the field is modified, the procedure must explicitly redisplay the field to view the effect of the DEBLANK attribute.

DEBUG( ) method

Starts and initializes the Debugger, and immediately gives control to the Debugger in stand-alone mode while blocking the invoking procedure.

Return type: LOGICAL

Applies to: DEBUGGER system handle

Syntax

```
DEBUG ( )
```

This method has the same effect as starting OpenEdge with the Debugger (`-debug`) startup parameter, except that instead of running the Debugger from the OpenEdge command line, it runs it from the invoking procedure. The invoking procedure then waits to continue execution until the Debugger exits. Although the Debugger has no control over the invoking procedure, it can control any other procedure started with the Debugger RUN option.

If the Debugger starts successfully, this method returns TRUE after the Debugger exits. Otherwise, it returns FALSE with no effect.

Note: To use this method, you must have the Application Debugger installed in your OpenEdge environment.

For more information about running the Debugger in stand-alone mode, see *Troubleshoot ABL Applications*. For more information about the DEBUGGER system handle, see DEBUGGER system handle.

DEBUG-ALERT attribute

Indicates whether the AVM provides access to ABL and .NET stack trace information, where applicable, when an error occurs during a session. Its primary use is for error alert boxes, so you can determine where an error occurred in your code.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

When an unhandled ABL error or .NET Exception occurs in an interactive session for any ABL client, the AVM displays an Error message box. When `DEBUG-ALERT` is TRUE, the message box also includes a **Help** button, which when pressed, produces a Stack Trace dialog box containing ABL and .NET stack trace information. If the error was generated by throwing an error object from the ABL code and the Error Stack (`-errorstack`) startup parameter is used, or the `ERROR-STACK-TRACE` attribute on the `SESSION` system handle is set to TRUE, then the stack trace also includes the contents of the `CallStack` property for the unhandled error. The ABL and .NET stack traces are also written to the client log when the Client Logging (`-clientlog`) startup parameter is specified. The top of the stack (most recent call) is displayed at the top of the trace listing.

You can also set `DEBUG-ALERT` to TRUE using the Debug Alert (`-debugalert`) startup parameter. For more information see *Startup Command and Parameter Reference*.

For more information about debugging from an alert dialog box, see *Troubleshoot ABL Applications*.

DECIMALS attribute

Indicates the number of decimal places, after the decimal point, that are stored for a buffer-field object that corresponds to a DECIMAL field. Valid values are zero (0) to ten (10). When `DECIMALS` is explicitly set (or is inherited from a database field via LIKE), the AVM rounds off any value that you assign via the `BUFFER-VALUE( )` method of a buffer field handle (or via the "::" syntax on a buffer handle) to the specified number of decimal places before completing the assignment. If DECIMALS is not set, the AVM rounds the value off to 10 decimal places if necessary.

If the WRITE-XML() method is used on the object containing the buffer field, then the XML Document generated will write the decimal field with the number of decimal places specified.

Note: The AVM determines the number of decimal places to display, as opposed to store, from the FORMAT attribute (not the DECIMALS attribute) of the buffer-field.

Data type: INTEGER

Access: Readable/Writeable; Read-only if the buffer-field object handle is for a database field

Applies to: Buffer-field object handle

DECLARE-NAMESPACE( ) method

Adds a namespace declaration to a tag in the XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
DECLARE-NAMESPACE ( namespace-URI [ , prefix ] )
```

namespace-URI

A CHARACTER or LONGCHAR expression evaluating to the URI of the attribute.

prefix

A CHARACTER or LONGCHAR expression evaluating to the prefix of the namespace.

Call this method to add a namespace declaration to a start tag. You can only call this method directly following a call to START-ELEMENT, WRITE-EMPTY-ELEMENT, INSERT-ATTRIBUTE, or DECLARE-NAMESPACE method. That is, you can only call this method when the WRITE-STATUS attribute is SAX-WRITE-TAG. The WRITE-STATUS attribute remains SAX-WRITE-TAG after this method call.

Regardless of the value of the STRICT attribute, this method fails if you do not call it after one of the valid methods listed above.

If you use an empty string (`""`) or the Unknown value (?) for the prefix, or you omit the prefix, then the method declares the default namespace: `xmlns="namespace-URI"`.

DEFAULT attribute

Indicates whether the button is a default button.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BUTTON widget

If the DEFAULT attribute is TRUE, the specified button is a default button. To make the specified button the default button for the frame, you must also set the frame's DEFAULT-BUTTON attribute to the handle of the button. A default button is one that handles all RETURN events when no other RETURN-enabling widget in the frame or dialog box has focus. RETURN-enabling widgets include any field-level widget for which a RETURN trigger is defined, or any button, whether or not it has a trigger defined. Thus, if a button has focus, that button handles the next RETURN event. If any other field-level widget without a RETURN trigger has focus, the default button handles the next RETURN event.

You can set this attribute only before the widget is realized.

Note: When the frame receives a default RETURN event, it actually sends a CHOOSE event to the default button. If the user presses the **RETURN** key while in a frame that has no default button, and the frame is part of a frame family, the AVM applies the CHOOSE event to the first default button it can find within the frame family in random order.

DEFAULT-BUFFER-HANDLE attribute

Like static temp-tables, every dynamic temp-table is created with at least one buffer. This buffer's object handle is returned by this attribute. DEFAULT-BUFFER-HANDLE cannot be called until the TEMP-TABLE-PREPARE( ) method has been called, since the default buffer is not created until then.

Data type: HANDLE

Access: Read-only

Applies to: Temp-table object handle

DEFAULT-BUTTON attribute

Indicates whether a button is a default button for the frame or dialog box.

Data type: HANDLE

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget

To make the specified button the default button for the frame or dialog box, you must also set the button's DEFAULT attribute to TRUE.

The DEFAULT-BUTTON attribute for frames is not supported in character mode.

The default button cannot display an image.

A default button is one that handles all RETURN events when no other RETURN-enabling widget in the frame or dialog box has focus. RETURN-enabling widgets include any field-level widget for which a RETURN trigger is defined, or any button, whether or not it has a trigger defined. Thus, if a button has focus, that button handles the next RETURN event. If any other field-level widget without a RETURN trigger has focus, the default button handles the next RETURN event.

Note: When the frame receives a default RETURN event, it actually sends a CHOOSE event to the default button. If the user presses the **RETURN** key while in a frame that has no default button, and the frame is part of a frame family, the AVM applies the CHOOSE event to the first default button it can find within the frame family in random order.

DEFAULT-COMMIT attribute
(PAS for OE only)

Indicates how an open transaction under the control of a transaction initiating procedure is to complete if the procedure is deleted in the absence of any SET-COMMIT( ) method or SET-ROLLBACK( ) method.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Transaction object handle

Setting the DEFAULT-COMMIT attribute to TRUE ensures that the transaction is completed. Setting it to FALSE ensures that the transaction is rolled back. The default value is FALSE.

Note: One common event that can terminate an open transaction is deleting the transaction initiating procedure that created the transaction.

DEFAULT-STRING attribute

The unformatted version of the INITIAL attribute.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer-field object handle

For example, consider the following temp-table definition:

```
DEFINE TEMP-TABLE ttFoo
  FIELD dFoo AS DATE    INITIAL TODAY
  FIELD iFoo AS INTEGER INITIAL 5000.
```

The DEFINE TEMP-TABLE statement defines the DATE field ttFoo.dFoo with an INITIAL value of TODAY. For this field, the INITIAL attribute is [today's date], formatted subject to SESSION:DATE-FORMAT. However, the DEFAULT-STRING attribute contains TODAY, the unformatted character value.

The INTEGER field ttFoo.iFoo is defined with an INITIAL value of 5000. The INITIAL attribute is the formatted 5,000. The DEFAULT-STRING attribute contains the unformatted 5000.

DEFAULT-STRING renders its dates and numeric values in LOCAL format, subject to the DATE-FORMAT and NUMERIC-FORMAT attributes of the SESSION system handle.

See also

DATE-FORMAT attribute, DEFAULT-VALUE attribute, INITIAL attribute, NUMERIC-FORMAT attribute

DEFAULT-VALUE attribute

The initial value of a table column in the native data type of the buffer-field.

Data type: The data type of the corresponding buffer-field

Access: Read-only

Applies to: Buffer-field object handle

For a DATE field with an INITIAL value of TODAY, DEFAULT-VALUE would contain a DATE value with today's date. For a DATETIME field with an INITIAL value of NOW, DEFAULT-VALUE would contain a DATETIME value with the current time. For a DECIMAL field, DEFAULT-VALUE would contain a DECIMAL value, as opposed to the INITIAL attribute, which has a CHARACTER data type.

DEFAULT-VALUE is suitable for assignment into a newly created record, but not as the initial value in the ADD-NEW-FIELD( ) method. In this case, TODAY and NOW will be the values and not the special strings for TODAY and NOW. That is, DEFAULT-VALUE for the initial argument in ADD-NEW-FIELD( ) will return INITIAL [today's date], instead of INITIAL TODAY.

In order for ADD-NEW-FIELD( ) to work correctly with an existing buffer for a new temp-table being dynamically built, you can use the ADD-LIKE-FIELD( ) method or, rather than using DEFAULT-VALUE, you can use the DEFAULT-STRING attribute, which is the unformatted, character data type version of the DEFAULT-VALUE including TODAY and NOW.

See also

ADD-LIKE-FIELD( ) method, ADD-NEW-FIELD( ) method, DATE-FORMAT attribute, DEFAULT-STRING attribute, INITIAL attribute, NUMERIC-FORMAT attribute

---

DELETE( ) method

Deletes an item from a combo box, radio-set, or selection list.

Return type: LOGICAL

Applies to: BROWSE widget (column), COMBO-BOX widget, RADIO-SET widget, SELECTION-LIST widget

Syntax (combo-box, selection-list, or browse column)

This is the syntax for a combo-box, selection-list, or browse column.

```
DELETE ( list-index | list-item )
```

list-index

An integer expression that specifies the ordinal position of a value in the combo box list or selection list.

list-item

A character-string expression that specifies a single value or a delimiter-separated list of values in the widget.

The DELETE( ) method removes the item specified by *list-index*, or removes the specified *list-item* from the list. *list-item* can represent multiple items. For example, you might specify `DELETE( "Chicago,Boston,New York" )`, where the delimiter is a comma. The delimiter is a comma by default or is specified by the DELIMITER attribute. If the method is successful, it returns TRUE.

For browses, this method applies only to combo-box browse columns.

Syntax (radio-set)

This is the syntax for a radio-set.

```
DELETE ( label )
```

label

A character-string expression that specifies an item to delete from the radio-set.

The DELETE( ) method deletes the item from the radio-set, whose appearance changes depending on the user interface and the setting of the AUTO-RESIZE attribute. For all user interfaces, if AUTO-RESIZE is TRUE, the remaining items collapse toward the top to fill the gap left by the deleted item. If AUTO-RESIZE is FALSE in Windows, the remaining items are repositioned to evenly span the original radio-set dimensions; in character interfaces, the remaining items collapse upward as when AUTO-RESIZE is TRUE.

If the method is successful, it returns TRUE.

Note: A single call to DELETE can delete one or more items from a combo box or selection list, or one item from a radio set.

DELETE-CHAR( ) method

Deletes the character at the current text cursor position.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
DELETE-CHAR ( )
```

If the character is successfully deleted, the method returns TRUE.

DELETE-CURRENT-ROW( ) method

Deletes the most recently selected row from a browse and the results list.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
DELETE-CURRENT-ROW ( )
```

This method does **not** delete the record from the database and has no effect on the database buffer. If you then want to delete the database record associated with the row, use the DELETE statement.

If the row is successfully deleted from the browse and results list, the method returns TRUE.

DELETE-HEADER-ENTRY( ) method

Deletes the XML underlying a SOAP-header-entryref object, without deleting the object.

Return type: LOGICAL

Applies to: SOAP-header-entryref object handle

Syntax

```
DELETE-HEADER-ENTRY ( )
```

DELETE-LINE( ) method

Deletes the line that currently contains the text cursor.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
DELETE-LINE ( )
```

If the line is successfully deleted, the method returns TRUE.

DELETE-NODE( ) method

Unlinks and deletes the node and its sub-tree from the XML document. The ABL handle is not deleted.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
DELETE-NODE( )
```

The following example demonstrates the use of the DELETE-NODE( ) method. Only use this when you are through using the node and all of its descendants.

```
hOldNode:DELETE-NODE( ).
```

DELETE-RESULT-LIST-ENTRY( ) method

Deletes the current row of a query's result list.

Return type: LOGICAL

Applies to: BROWSE widget, Query object handle

Syntax

```
DELETE-RESULT-LIST-ENTRY ( )
```

For the browse, DELETE-RESULT-LIST-ENTRY( ) solves the following problem: Suppose you create a browse with a primary table and a secondary table, and in the primary table, the key to the secondary table changes. The AVM never displays the new secondary table because the result list entry contains the rowid of the original secondary table.

When you use DELETE-RESULT-LIST-ENTRY( ) together with CREATE-RESULT-LIST-ENTRY( ), you can update the result list entry and display the modified row without having to reopen the query.

For example, suppose you create a browse with customer.name, customer.salesrep, and salesrep.repname (from the Sports2020 database). Then, in one record of the browse, you change customer.salesrep from "bbb" to "dkp."

Without using DELETE-RESULT-LIST-ENTRY( ), the secondary record remains "bbb" until the query is reopened.

The following code fragment uses DELETE-RESULT-LIST-ENTRY( ) and CREATE-RESULT-LIST-ENTRY( ) to display the modified secondary record:

```
ON ROW-LEAVE OF my-brow DO:
  DEFINE VARIABLE num AS INTEGER NO-UNDO.
  DEFINE VARIABLE ok  AS LOGICAL NO-UNDO.

  IF Customer.SalesRep:MODIFIED THEN DO:
    num = Customer.CustNum.
    ok  = my-brow:DELETE-RESULT-LIST-ENTRY().

    /* DELETE-RESULT-LIST-ENTRY() disconnects recs from rec bufs, so re-read
       Customer and SalesRep records */
    FIND Customer WHERE Customer.CustNum EQ num.
    FIND SalesRep WHERE Salesrep.SalesRep EQ Customer.SalesRep:SCREEN-VALUE.
    /* Create new result list entry with "new" secondary table's rowid */
    ok = my-brow:CREATE-RESULT-LIST-ENTRY().
    /* Update viewport */
    DISPLAY SalesRep.RepName WITH BROWSE my-brow.
  END.
END. /* ROW-LEAVE OF my-brow */
```

Note: During this operation, the query pointer must not move.

DELETE-SELECTED-ROW( ) method

Deletes the *n*th selected row from a browse and the results list.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
DELETE-SELECTED-ROW ( n )
```

n

An integer expression that specifies a selected row within the browse.

Note: Do not confuse the DELETE-SELECTED-ROW method (note the singular) with the DELETE-SELECTED-ROWS method (note the plural).

The AVM maintains a numbered list of selected rows, starting at 1. When the DELETE-SELECTED-ROW(*n*) method is encountered, the AVM searches this list to find the *n*th selected row.

This method does **not** delete the record from the database and has no effect on the database buffer. If you want to delete the database record associated with the row, use the DELETE statement.

If the row is successfully deleted, the method returns TRUE.

If you want to delete all selected rows, whether it is one or many, DELETE-SELECTED-ROWS is the preferred, optimized method for doing so.

DELETE-SELECTED-ROWS( ) method

Deletes all currently selected rows from a browse and the associated results list.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
DELETE-SELECTED-ROW ( )
```

Note: Do not confuse the DELETE-SELECTED-ROW method (note the singular) with the DELETE-SELECTED-ROWS method (note the plural).

This method does **not** delete the record from the database and has no effect on the database buffer. If you want to delete the database record associated with the row, use the DELETE statement.

If the row is successfully deleted, the method returns TRUE.

DELIMITER attribute

The character that separates values input to or output from a combo box or selection list.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget (column), COMBO-BOX widget, RADIO-SET widget, SELECTION-LIST widget

Delimiter character can have any ASCII value from 1 to 127. The default delimiter is a comma.

For browses, this attribute applies only to combo-box browse columns.

DESCRIPTION attribute

A character description of the current profiling session. This string is included as part of the data that is written to the profile output file. The default value is the unknown value (?).

Data type: CHARACTER

Access: Readable/Writable

Applies to: PROFILER system handle

DESELECT-FOCUSED-ROW( ) method

Deselects the row with current focus.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
DESELECT-FOCUSED-ROW ( )
```

This method is ignored on single-select browse widgets, because focus follows selection.

DESELECT-ROWS( ) method

Deselects all currently selected rows in the browse and clears the associated record buffer.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
DESELECT-ROWS ( )
```

If the rows are successfully deselected, the method returns TRUE.

DESELECT-SELECTED-ROW( ) method

Deselects the *n*th selected row in a browse.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
DESELECT-SELECTED-ROW ( n )
```

n

An integer expression that specifies a selected row within the browse.

The AVM maintains a numbered list of selected rows, starting at 1. When the DESELECT-SELECTED-ROW(*n*) method is encountered, the AVM searches this list to find the *n*th selected row. If the row is successfully deselected, the method returns TRUE.

DETACH-DATA-SOURCE( ) method

Detaches a data-source object from a temp-table buffer in a ProDataSet object.

Return type: LOGICAL

Applies to: Buffer object handle

See also

ATTACH-DATA-SOURCE( ) method

DIRECTORY attribute

The directory to which the output file is stored. The attribute cannot be set to the unknown value (?), and only accepts a valid directory. The default value is either the `-T` directory or the current working directory if `-T` was not specified.

Data type: CHARACTER

Access: Readable/Writable

Applies to: PROFILER system handle

DISABLE( ) method

Disables the radio set button. The selected radio-set button's SCREEN-VALUE is changed to the first enabled radio set button or to the Unknown value (?) if all radio set buttons are disabled.

Return type: LOGICAL

Applies to: RADIO-SET widget

Syntax

```
DISABLE ( label )
```

label

A character-string expression that specifies the label of a button in the radio set.

If the operation is successful, the method returns TRUE.

DISABLE-AUTO-ZAP attribute

Indicates whether the AVM ignores the value of the AUTO-ZAP attribute.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget (column), COMBO-BOX widget, FILL-IN widget

If DISABLE-AUTO-ZAP is TRUE, the AVM ignores the value of the AUTO-ZAP attribute and assumes it is FALSE. If the DISABLE-AUTO-ZAP attribute is FALSE, the AVM assumes the value of the AUTO-ZAP attribute is TRUE.

DISABLE-CONNECTIONS( ) method

Indicates that the AVM no longer listen for or accept new connections on the port associated with the server socket. However, all existing connections are still valid.

Return type: LOGICAL

Applies to: Server socket object handle

Syntax

```
DISABLE-CONNECTIONS( )
```

Returns TRUE if connections are disabled for this server socket object, even if the server socket object was never enabled or was already disabled.

DISABLE-DUMP-TRIGGERS( ) method

Allows a user to access a buffer object's table without firing FIND triggers.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
DISABLE-DUMP-TRIGGERS ( )
```

To run this method, the end user must have CAN-DUMP privileges for the table.

Triggers cannot be disabled from a persistent procedure.

The FIND trigger remains disabled until the procedure in which it is disabled returns.

See also

DISABLE TRIGGERS statement

DISABLE-LOAD-TRIGGERS( ) method

Allows you to create or update a buffer object's table without executing update triggers such as CREATE, WRITE, DELETE or ASSIGN.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
DISABLE-LOAD-TRIGGERS ( allow-replication )
```

allow-replication

A LOGICAL expression indicating whether to disable or allow replication triggers. When TRUE, replication triggers are allowed to run, including the DELETE trigger. When FALSE, all triggers are disabled.

To run this method, you must have CAN-LOAD privileges for the table.

Triggers cannot be loaded from a persistent procedure.

The triggers remain disabled until the procedure in which it is disabled returns.

Note: The Disable Delete Trigger (`-disabledeltrig`) startup parameter disables the DELETE trigger when ALLOW-REPLICATION is on for the DISABLE TRIGGERS statement or the DISABLE-LOAD-TRIGGER() buffer method. For more information about this parameter, see *Startup Command and Parameter Reference*.

See also

DISABLE TRIGGERS statement

DISCONNECT( ) method (Handle)

Disconnects the client from the application server or Web service currently associated with the specified server handle. For the socket, closes the socket by terminating the connection between the socket and the port to which it is connected.

Return type: LOGICAL

Applies to: Server object handle, Server socket object handle

Syntax

```
DISCONNECT ( )
```

When the application server receives a disconnect request:

- **For a session-managed application server** — Control returns immediately to the client application, and any configured Disconnect procedure executes in the connected application server session.
- **For a state-free application server** — Any configured Disconnect procedure executes in an available application server session, then control returns to the client application.
- **For any application server with pending asynchronous requests**— All running or pending asynchronous requests are cancelled and the associated event procedure or method is called for each request. The CANCELLED attribute on the asynchronous request handle for all such cancelled requests is set to TRUE.

If an error occurs during the disconnection from an application server, DISCONNECT( ) returns FALSE. An error occurs if:

- The server handle is invalid.
- The server is either not connected or already disconnected.

If DISCONNECT( ) completes successfully, the CONNECTED( ) method returns FALSE. In addition, all attributes of the SERVER object (except for the FIRST PROCEDURE and LAST PROCEDURE attributes) are restored to their initial state. Specifically, SUBTYPE returns the Unknown value (?).

For sockets, the AVM automatically closes a socket if it detects that the connection to which the socket is bound has failed or been terminated.

DISPLAY-MESSAGE( ) method

Displays a message in an alert box. The Debugger stores these messages and displays them to the user in an alert box when the Debugger regains control of an application.

Return type: INTEGER

Applies to: DEBUGGER system handle

Syntax

```
DISPLAY-MESSAGE ( char-expression )
```

char-expression

Any character expression.

This method appends a new line to *char-expression* before displaying the specified string. If the Debugger is initialized and *char-expression* is a valid character expression, this method returns TRUE. Otherwise, it returns FALSE with no effect.

Note: To use this method, you must have the Application Debugger installed in your OpenEdge environment.

DISPLAY-TIMEZONE attribute

The time zone offset, in minutes, used to display DATETIME-TZ data. The default value is the session's time zone offset.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SESSION system handle

If the format string for the DATETIME-TZ does not include the time zone offset, the AVM converts the DATETIME-TZ data to the local date and time and displays the data with the time zone offset specified by this attribute.

If the format string for the DATETIME-TZ includes the time zone offset, the AVM ignores this attribute and displays the data in the local time of the value, along with the time zone offset.

Set this attribute to the Unknown value (?) to use the session's time zone offset for display.

DISPLAY-TYPE attribute

The type of display for an r-code file or session.

Data type: CHARACTER

Access: Read-only

Applies to: RCODE-INFO handle, SESSION system handle

With the RCODE-INFO handle, the DISPLAY-TYPE attribute holds a value of "GUI" for a graphical display, "TTY" for a character-mode display, or a blank string ("") for code that does not contain any display-specific statements. Use of this attribute allows developers to determine if the r-code contains statements that make it executable only in the same display environment on which it was compiled. If r-code contains ABL statements that interact with the UI (such as DISPLAY), it can only be executed in the same UI. The same restriction does not apply to r-code that does not contain UI-specific statements.

With the SESSION handle, the DISPLAY-TYPE attribute holds a value of "GUI" for a graphical display or "TTY" for a character-mode display, depending on the display type of the running session.

DOMAIN-DESCRIPTION attribute

The description of the domain in which the user represented by the client-principal object was authenticated. If not specified, the AVM returns a blank ("") character string.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

If not already assigned, during any OpenEdge-performed user authentication operation, OpenEdge assigns the value of this attribute (if any) from the domain configuration used to authenticate the asserted identity before sealing the client-principal object.

Once the client-principal object is sealed, this attribute is read-only, and attempting to write to it raises a run-time error.

See also

DOMAIN-NAME attribute, DOMAIN-TYPE attribute

DOMAIN-NAME attribute

The name of the OpenEdge domain in which to authenticate the user identity represented by the unsealed client-principal object. This domain is also configured with an access code used to validate the same identity after the client-principal has been sealed. You must set this attribute before you can seal the associated client-principal object with the:

- SEAL( ) method as part of an application-performed user authentication operation
- SET-CLIENT( ) method or SET-DB-CLIENT function in an OpenEdge-performed user authentication operation

The domain name you specify must match a domain entry registered in the trusted domain registry. Along with the domain access code, this entry identifies the authentication system used to validate the user's identity for user authentication and single sign-on (SSO) operations and specifies the user's tenancy for multi-tenant database connections in which their identity is set. For more information on domains, authentication systems, and database tenancy, see *Learn about Identity Management*.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

You can assign this attribute the following values:

- Empty string ("") — The default blank domain name
- Non-empty, case-insensitive string — A maximum of 64 characters from the following restricted character set:
  - a to z
  - A to Z

- 0 to 9
- Any of the following seven special characters:

```
# $ % & - _ .
```

This attribute can be set using the INITIALIZE( ) method or the QUALIFIED-USER-ID attribute. Writing this attribute directly overwrites any value previously set by any mechanism.

If the attribute has never been initialized, reading its value returns the empty string (""). Otherwise, it returns the most recent setting using the INITIALIZE( ) method, QUALIFIED-USER-ID attribute, or the DOMAIN-NAME attribute, itself.

Once the client-principal object is sealed, this attribute is read-only, and attempting to write to it raises a run-time error.

ABL also raises ERROR if you attempt to assign:

- The Unknown value (?)
- Any character expression that does not conform to domain name length and character set restrictions
- A character expression when the client-principal object is sealed and in the LOGIN, LOGOUT, EXPIRED, or FAILED state (see the LOGIN-STATE attribute entry)

See also

DOMAIN-DESCRIPTION attribute, DOMAIN-TYPE attribute, INITIALIZE( ) method, QUALIFIED-USER-ID attribute, USER-ID attribute

DOMAIN-TYPE attribute

The name of the authentication system that is configured for the domain specified by the DOMAIN-NAME attribute. This authentication system authenticates or validates the user identity represented by the client-principal object.

Note: The name assigned to this attribute is for information purposes only and does not participate in authentication operations. OpenEdge always uses the name in the domain configuration itself to identify the authentication system to use.

The configured authentication system determines the operation type (user authentication, single sign-on (SSO), or both) and mechanism (OpenEdge-defined or application-defined) that is available to validate and set a given user identity. For more information on authentication systems, see *Learn about Identity Management*.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

If the attribute has not been set, reading it returns the empty string ("").

If not already assigned, during any OpenEdge-performed user authentication operation, OpenEdge assigns the value of this attribute from the domain configuration used to authenticate the asserted identity before sealing the client-principal object.

Once the client-principal object is sealed, this attribute is read-only, and attempting to write to it raises a run-time error.

See also

DOMAIN-DESCRIPTION attribute, DOMAIN-NAME attribute

DOWN attribute

Indicates the number of iterations in a down frame that contain data or number of potential rows in a browse widget. For a one-down frame, the value of DOWN is 1. Sets the number of browse-rows that appear in the viewport.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget, FRAME widget

In character mode, the DOWN attribute must be set before the browse is realized. Attempting to set the attribute after realization of the browse causes a run-time error.

If you change the value of a browse's DOWN attribute, you change the number of rows that appear in the viewport, which might change the value of the browse's HEIGHT-CHARS and HEIGHT-PIXELS attributes. Changing the value of a browse's DOWN attribute does not change the value of the browse's ROW-HEIGHT-CHARS and ROW-HEIGHT-PIXELS attributes.

Note: If the browse's height is set with the DOWN attribute and a browse column is added using the ADD-CALC-COLUMN( ), ADD-COLUMNS-FROM( ) or ADD-LIKE-COLUMN( ) methods, the browse's height may change to ensure that the number of DOWN is preserved.

DRAG-ENABLED attribute

Indicates whether the user can simultaneously hold down the mouse select button and drag the mouse cursor through the selection list. As the mouse cursor passes over an item, the item is highlighted. When the user releases the select button, the highlighted item becomes the selected item.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SELECTION-LIST widget

In this style of selecting, a user can deselect an item only by selecting another. Once an item is selected, the list cannot revert to its unselected state.

The default value for this attribute is TRUE.

Note: In Windows, DRAG-ENABLED is always TRUE.

You can set this attribute only before the widget is realized.

DROP-TARGET attribute
(Windows only; Graphical interfaces only)

Indicates whether the widget can accept dropped files.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

If DROP-TARGET is TRUE, dragging one or more files over the widget causes the mouse pointer to change to indicate that the widget can accept the files. The default value of DROP-TARGET is FALSE.

For related information, see the reference entries for the DROP-TARGET option of the DEFINE BROWSE statement, DEFINE BUTTON statement, DEFINE FRAME statement, and DEFINE VARIABLE statement.

DUMP-LOGGING-NOW( ) method

Writes the accumulated query statistics to the log file for the query associated with a query handle.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
DUMP-LOGGING-NOW ( reset-expression )
```

reset-expression

A logical expression where TRUE indicates that the AVM clear the query statistics after writing the statistics to the log file, and FALSE indicates that the AVM leave the query statistics unchanged.

If the AVM writes the query statistics to the log file successfully, the method returns TRUE. If the AVM does not write the query statistics to the log file, the method returns FALSE. The AVM does not write query statistics to the log file under the following conditions:

- QryInfo logging for the specified query was not turned on before the query started.

  To turn on basic logging for an individual query, you must set the BASIC-LOGGING attribute to TRUE before a query starts. For a dynamically opened query, this is before the QUERY-PREPARE( ) method. For a statically opened query, this is before the OPEN QUERY statement.

- QryInfo logging was turned off for the specified query, or all queries, before the query completed.

You can use the DUMP-LOGGING-NOW( ) method only when the logging level of the QryInfo log entry type is set to level 2 (Basic) or higher. To set the logging level of the QryInfo log entry type, use the LOG-ENTRY-TYPES attribute or the Log Entry Types (`-logentrytypes`) startup parameter with the logging level option.

If the logging level of the QryInfo log entry type is set to level 2 (Basic), the DUMP-LOGGING-NOW( ) method writes query statistics to the log file only if the BASIC-LOGGING attribute is set to TRUE before the query started.

For more information about the Log Entry Types (`-logentrytypes`) startup parameter, see *Startup Command and Parameter Reference*.

For more information about logging query statistics, see *Troubleshoot ABL Applications*.

DYNAMIC attribute

Indicates whether the widget is dynamic or static.

Data type: LOGICAL

Access: Read-only

Applies to: BROWSE widget, Buffer object handle, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, ProDataSet object handle, Query object handle, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, Temp-table object handle, TEXT widget, TOGGLE-BOX widget, WINDOW widget

If the DYNAMIC attribute is TRUE, the widget is dynamic, otherwise it is static.

EDGE-CHARS attribute

The width, in character units, of the edge of a rectangle.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: RECTANGLE widget

EDGE-PIXELS attribute

The width, in pixels, of the edge of a rectangle.

Data type: INTEGER

Access: Readable/Writeable

Applies to: RECTANGLE widget

EDIT-CAN-PASTE attribute
(Windows only; Graphical interfaces only)

Indicates whether the Clipboard contains data that can be pasted into the widget.

Data type: LOGICAL

Access: Read-only

Applies to: BROWSE widget (cell), COMBO-BOX widget, EDITOR widget, FILL-IN widget

If the Clipboard contains data that can be pasted into the widget, EDIT-CAN-PASTE is TRUE. Otherwise, it is FALSE.

For browses, this attribute applies only to fill-in and combo-box browse columns.

EDIT-CAN-UNDO attribute
(Windows only; Graphical interfaces only)

Indicates whether the widget can undo the last modification.

Data type: LOGICAL

Access: Read-only

Applies to: BROWSE widget (cell), COMBO-BOX widget, EDITOR widget

If the widget can undo the last modification, EDIT-CAN-UNDO has the value TRUE, otherwise it has the value FALSE.

If you set EDIT-CAN-UNDO to any value, the AVM empties the undo buffer.

For browses, this attribute applies only to fill-in and combo-box browse columns.

EDIT-CLEAR( ) method
(Windows only; Graphical interfaces only)

Deletes the selected text.

Return type: LOGICAL

Applies to: BROWSE widget (cell), COMBO-BOX widget, EDITOR widget, FILL-IN widget

Syntax

```
EDIT-CLEAR ( )
```

If the widget performs the operation successfully, the method returns TRUE. Otherwise, it returns FALSE.

For browses, this method applies only to fill-in and combo-box browse columns.

EDIT-COPY( ) method
(Windows only; Graphical interfaces only)

Copies the currently selected text in the widget to the Clipboard.

Return type: LOGICAL

Applies to: BROWSE widget (cell), COMBO-BOX widget, EDITOR widget, FILL-IN widget

Syntax

```
EDIT-COPY ( )
```

If the widget performs the operation successfully, the method returns TRUE. Otherwise, it returns FALSE.

For browses, this method applies only to fill-in and combo-box browse columns.

EDIT-CUT( ) method
(Windows only; Graphical interfaces only)

Copies the currently selected text in the widget to the Clipboard and then deletes the selected text.

Return type: LOGICAL

Applies to: BROWSE widget (cell), COMBO-BOX widget, EDITOR widget, FILL-IN widget

Syntax

```
EDIT-CUT ( )
```

If the widget performs the operation successfully, the method returns TRUE. Otherwise, it returns FALSE.

For browses, this method applies only to fill-in and combo-box browse columns.

EDIT-PASTE( ) method
(Windows only; Graphical interfaces only)

Copies the currently selected text of the Clipboard into the widget at the current cursor position, if the Clipboard contains text data.

Return type: LOGICAL

Applies to: BROWSE widget (cell), COMBO-BOX widget, EDITOR widget, FILL-IN widget

Syntax

```
EDIT-PASTE ( )
```

If the widget performs the operation successfully, the method returns TRUE. Otherwise, it returns FALSE.

For browses, this method applies only to fill-in and combo-box browse columns.

EDIT-UNDO( ) method
(Windows only; Graphical interfaces only)

Makes the editor undo its most recent edit if possible.

Return type: LOGICAL

Applies to: BROWSE widget (cell), COMBO-BOX widget, EDITOR widget

Syntax

```
EDIT-UNDO ( )
```

If the widget performs the operation successfully, the method returns TRUE. Otherwise, it returns FALSE.

For browses, this method applies only to fill-in and combo-box browse columns.

EMPTY attribute

Indicates whether the SCREEN-VALUE attribute for the editor contains text.

Data type: LOGICAL

Access: Read-only

Applies to: EDITOR widget

The EMPTY attribute is TRUE if the editor contains no text (that is, the editor's SCREEN-VALUE is null).

EMPTY-DATASET( ) method

Empties a ProDataSet object of all records in its associated temp-tables.

Return type: LOGICAL

Applies to: ProDataSet object handle

Syntax

```
EMPTY-DATASET ( )
```

When you invoke this method, the ABL Virtual Machine (AVM) sets the ORIGIN-HANDLE attribute on any changes temp-tables corresponding to the source temp-tables of this ProDataSet to the Unknown value (?).

EMPTY-TEMP-TABLE( ) method

Deletes all records from a temp-table associated with a buffer object or from a temp-table directly identified by a temp-table object handle.

When you empty a temp-table that is defined as UNDO within a transaction, the AVM deletes the records individually. This is less efficient than emptying the temp-table outside the transaction, where the AVM deletes all records in the temp-table as a unit.

Return type: LOGICAL

Applies to: Buffer object handle, Temp-table object handle

Syntax

```
EMPTY-TEMP-TABLE ( )
```

If the temp-table you invoke this method on was used to populate one or more temp-tables via the GET-CHANGES( ) method, the AVM sets the ORIGIN-HANDLE attribute for those populated temp-tables to the Unknown value (?).

Note: This method corresponds to the EMPTY TEMP-TABLE statement.

ENABLE( ) method

Enables the specified radio button within the radio set.

Return type: LOGICAL

Applies to: RADIO-SET widget

Syntax

```
ENABLE ( label )
```

label

A character-string expression that specifies the label of a radio button.

If the operation is successful, the method returns TRUE.

ENABLE-CONNECTIONS( ) method

Specifies the TCP/IP port that the AVM uses to listen for new connections. Once called, the AVM automatically listens for and accepts new connections for the specified port.

Return type: LOGICAL

Applies to: Server socket object handle

Syntax

```
ENABLE-CONNECTIONS ( connection-parms )
```

connection-parms

A character string expression that contains a space-separated list of one or more socket connection parameters.

The following table describes the socket connection parameters you can include in this string.

Table 76: Socket connection parameters

Parameter	Description
-S *socket-port*	The TCP/IP port number that the AVM should listen to and accept connections on. You can specify either an explicit port number or a TCP service name. If you use a TCP service name, the method uses the port number associated with that name in the TCP/IP services file.
-pf *filename*	Optional. A text file containing any of the socket connection parameters described in this table. If this file contains any other OpenEdge startup parameters, this method ignores them.
-qsize *backlog*	Optional. The length of the pending-connection queue—that is, the maximum number of connection requests you want the server to queue while it processes the connections already accepted. If the queue is full when a connection request is received, it is refused.The default length of the queue depends on the platform. **Note:** On some platforms, the value you supply for *backlog* is modified by addition, subtraction, multiplication, division, or some combination of these, and it is this modified value that becomes the maximum length of the queue. For more information, see the documentation for your platform.
-ssl	If specified, the connection is TLS-based. **Note:** Be sure you need TLS before using this option. TLS incurs more or less heavy performance penalties, depending on resources and load.
-keyalias *aliasname*	Sets the alias name of the Public[/Private] key and digital certificate to use within the keystore. If not specified, the default `default_server` key alias is used.
-keyaliaspasswd *encpwd*	Sets the password to use in accessing the Public[/Private] key and digital certificate. Always specify a `-keyaliaspassword` when the `-keyalias` option is used. The default password only exists when using the `default_server` server certificate.
-nosessioncache	If specified, caching for the TLS client session is disabled.
-sessiontimeout [*seconds*]	The maximum amount of time, in seconds, that the server waits before it rejects a TLS client's request to resume a session. The default value is 180 seconds.

An application server cannot act as a socket server, since they are already listening on a port. ENABLE-CONNECTIONS is only valid from batch clients, GUI clients and character clients. This method will generate an error if it is called from an invalid application. This method will also generate an error if it is called multiple times without the DISABLE-CONNECTION( ) method being called in between.

Note: Connections to a TLS-enabled server socket require the management of public keys on the client (TLS client) and private keys on the server (TLS server). For ABL sockets, the TLS client is the ABL session initiating the TLS connection on a socket object and the TLS server is the ABL session enabling TLS connections on a server socket object. For information on using TLS to secure an ABL socket connection, see the sections on sockets in *OpenEdge Programming Interfaces*. For more information on TLS and managing private key and digital certificate stores for OpenEdge TLS clients and servers, see *Learn about Security and Auditing*.

ENABLE-EVENTS( ) method
(Windows only)

Enables event notification for automation objects.

Return type: CHARACTER

Applies to: Automation Object

Syntax

```
ENABLE-EVENTS ( event-proc-prefix )
```

event-proc-prefix

A character-string expression that the AVM prepends to event names. The resulting string is the name of the internal procedure the AVM runs when an event is fired. During an event notification, all running procedures and all persistent procedures are searched to find a procedure with the name matching *event-proc-prefix.eventname* (for example, `ExcelWB.SelectionChanged`).

ENABLED attribute

Determines whether or not the profiler is enabled. When set to TRUE, the profiler is activated and initialized. The default value is FALSE.

Data type: LOGICAL

Access: Readable/Writable

Applies to: PROFILER system handle

When the profiler is first enabled, all the active procedures are registered and assigned module identifiers. An active procedure is any persistent procedure or any procedure on the procedure call stack. For each active procedure, except those that are part of Developer Studio, a PROFILER:LISTINGS setting of TRUE also generates listing files. In addition, a TRUE setting records coverage analysis information. To generate listing files or coverage analysis information, set PROFILER:LISTINGS and PROFILER:COVERAGE attributes to TRUE before setting PROFILER:ENABLED to TRUE. As long as profiling is enabled, Progress maintains a registry of the procedures that executed, regardless of whether profiling is turned on or off (controlled by the setting of the PROFILER:PROFILING attribute). When profiling is disabled (by setting PROFILER:ENABLED to FALSE), Progress writes accumulated profiling data to the output file, and then discards the registry of procedures that executed.

ENCODE-DOMAIN-ACCESS-CODE( ) method

Encrypts the specified character expression and returns a character value that you can store for later use as a domain access code in your database's domain configuration.

Return-type: CHARACTER

Applies to: SECURITY-POLICY system handle

Syntax

```
ENCODE-DOMAIN-ACCESS-CODE ( data-to-encrypt )
```

data-to-encrypt

A CHARACTER expression containing the domain access code.

See also

VALIDATE-DOMAIN-ACCESS-CODE( ) method

ENCODING attribute

Returns the name of the character encoding used to encode the contents of an XML document. The default encoding is UTF-8.

Data type: CHARACTER

Access: Readable/Writable

Applies to: X-document object handle, SAX-writer object handle

For an X-document object, the AVM sets the ENCODING attribute to the encoding name specified in the XML document's encoding declaration when you load an XML document using the LOAD( ) method (Handle).

You can also set the ENCODING attribute to the name of the character encoding to use when saving an XML document using the SAVE( ) method. The AVM records this character encoding in the encoding declaration in the XML document's prologue. If you do not set the ENCODING attribute, when you save the document, the document will not have an encoding declaration in its prologue, but the document will be saved with the default encoding of UTF-8.

For a SAX-writer object, you can set the ENCODING attribute to the name of the character encoding to use when writing the XML document. You can set this attribute only when the WRITE-STATUS is either SAX-WRITE-IDLE or SAX-WRITE-COMPLETE. That is, you can only change this attribute when the writer is not writing, otherwise it fails and generates an error message. The AVM records this character encoding in the encoding declaration in the XML document's prologue. If you do not set the ENCODING attribute, when you write the document, the document will not have an encoding declaration in its prologue, but the document will be written with the default encoding of UTF-8.

The encoding name must be an Internet Assigned Numbers Authority (IANA) name supported by the ABL XML Parser. The following table lists the names of the supported IANA encodings and their corresponding ABL code pages.

Table 77: IANA encodings and corresponding ABL code pages

IANA encoding name	ABL code page name
Big5	BIG-5
EUC-JP	EUCJIS
GB2312	GB2312
GB18030	GB18030
GBK	CP936
hp-roman8	ROMAN-8
IBM00858	IBM858
IBM037	IBM037
IBM273	IBM273
IBM277	IBM277
IBM278	IBM278
IBM284	IBM284
IBM297	IBM297
IBM437	IBM437
IBM500	IBM500
IBM850	IBM850
IBM851	IBM851
IBM852	IBM852
IBM857	IBM857
IBM861	IBM861

IANA encoding name	ABL code page name
IBM862	IBM862
IBM866	IBM866
ISO-8859-1	ISO8859-1
ISO-8859-2	ISO8859-2
ISO-8859-3	ISO8859-3
ISO-8859-4	ISO8859-4
ISO-8859-5	ISO8859-5
ISO-8859-6	ISO8859-6
ISO-8859-7	ISO8859-7
ISO-8859-8	ISO8859-8
ISO-8859-9	ISO8859-9
ISO-8859-10	ISO8859-10
ISO-8859-15	ISO8859-15
KOI8-R	KOI8-R
KS_C_5601-1987	KSC5601
Shift_JIS	SHIFT-JIS
TIS-620	620-2533
US-ASCII	-cpinternal
UTF-16	UTF-16
UTF-32	UTF-32
UTF-8	UTF-8
windows-1250	1250
windows-1251	1251
windows-1252	1252
windows-1253	1253
windows-1254	1254

IANA encoding name	ABL code page name
windows-1255	1255
windows-1256	1256
windows-1257	1257
windows-1258	1258

ENCRYPT-AUDIT-MAC-KEY( ) method

Encrypts and encodes the specified character expression and returns an encrypted character value that you can store for later use in message authentication code (MAC) operations.

Return type: CHARACTER

Applies to: AUDIT-POLICY system handle

Syntax

```
ENCRYPT-AUDIT-MAC-KEY ( encrypt-key )
```

encrypt-key

A character expression containing the key to encrypt. The AVM converts this key to UTF-8 before encrypting it and storing it, which ensures a consistent value regardless of code page settings.

The following code fragment illustrates how to use the ENCRYPT-AUDIT-MAC-KEY( ) method to set a database MAC key value:

```
DEFINE VARIABLE cVal AS CHARACTER NO-UNDO.
DEFINE VARIABLE cKey AS CHARACTER NO-UNDO INITIAL "Open Sesame".
. . .
cVal = AUDIT-POLICY:ENCRYPT-AUDIT-MAC-KEY(cKey).
FIND _db-detail.
PUT-STRING(_db-detail._db-mac-key, 1, LENGTH(cVal)) = cVal.
```

You can also use the ENCRYPT-AUDIT-MAC-KEY( ) method to generate an encrypted value for an OpenEdge password in order to obscure its cleartext value from hacking. For example:

```
DEFINE INPUT PARAMETER cUserID AS CHARACTER NO-UNDO.
DEFINE INPUT PARAMETER cPasswd AS CHARACTER NO-UNDO.

CONNECT C:\OpenEdge\WRK\db\Sports2020
  VALUE( "-U " + cUserID +
         " -P " + "oech1::" + AUDIT-POLICY:ENCRYPT-AUDIT-MAC-KEY(cPasswd))
  -H dbserver -S 1900 NO-ERROR.
```

The `cUserID` and `cPasswd` parameters pass in a user ID and password that a user might enter in response to a prompt, in this case to authenticate and set the user identity for a connection created for the `sports2020` database.

Therefore, `cPasswd` contains the cleartext value of the password the user has typed. The `"oech1::"` prefix is an identifier that tells OpenEdge that the value following the `"::"` is encrypted in a manner that allows it to recover the cleartext value for internal operations, such as validating a user account password. Without the prefix, OpenEdge interprets the string as the cleartext value a user might type directly, and therefore does not attempt to decrypt.

In this example, the encrypted password is passed as the value of the Password (`-P`) connection parameter. However, OpenEdge understands this encryption format anywhere that it accepts a password—for example, to set the PRIMARY-PASSPHRASE attribute on the client-principal object handle for OpenEdge-performed user authentication.

For more information on using password encryption in ABL, see the documentation on encrypted passwords in *OpenEdge Programming Interfaces*.

ENCRYPTION-SALT attribute

The default salt value (a random series of bytes) to use with the GENERATE-PBE-KEY function. The default value is the Unknown value (?), which indicates that no salt value is used to generate the password-based encryption key.

Data type: RAW

Access: Readable/Writeable

Applies to: SECURITY-POLICY system handle

If specified, this salt value is combined with a password value and hashed some number of times to generate a password-based encryption key (using the algorithm specified by the PBE-HASH-ALGORITHM attribute and the number of iterations specified by the PBE-KEY-ROUNDS attribute).

When set, only the first 8 bytes are used. If the value has fewer than 8 bytes, it is padded at the end with zero-value bytes.

You can use the GENERATE-PBE-SALT function to generate a salt value, which can help to ensure that the password key value is unique.

You are responsible for generating, storing, and transporting this value.

See also

GENERATE-PBE-KEY function, GENERATE-PBE-SALT function

END-DOCUMENT( ) method

Closes the XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
END-DOCUMENT( )
```

Closes the XML stream. This is the logical conclusion of creating the XML document.

If you call this method before the START-DOCUMENT method, the method fails. WRITE-STATUS must not be SAX-WRITE-IDLE or SAX-WRITE-COMPLETE when you call this method. END-DOCUMENT changes WRITE-STATUS to SAX-WRITE-COMPLETE.

If the STRICT attribute is TRUE and the final tag has not been closed (that is, the root node), then this method fails.

See also

START-DOCUMENT( ) method

END-ELEMENT( ) method

Ends an XML node based upon the name of the node with in a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
END-ELEMENT ( name [, namespace-URI] )
```

name

A CHARACTER or LONGCHAR expression evaluating to the fully qualified or unqualified name of the element.

namespace-URI

A CHARACTER or LONGCHAR expression evaluating to the URI of the element. If the element doesn't contain a namespace, it can evaluate to an empty string ("") or the Unknown value (?).

Ends an XML node and sets the WRITE-STATUS to SAX-WRITE-ELEMENT.

For every invocation of END-ELEMENT, there must be a preceding corresponding call of the START-ELEMENT method. All the parameter values must match for the methods to correspond. The method does not resolve namespaces. Instead, it matches the namespace against the corresponding START-ELEMENT value.

If the STRICT attribute is TRUE and the method does not match a preceding START-ELEMENT call, then the method fails.

See also

START-ELEMENT( ) method

END-EVENT-GROUP( ) method

Indicates the end of a group of related events in the current session.

Return type: LOGICAL

Applies to: AUDIT-CONTROL system handle

Syntax

```
END-EVENT-GROUP( )
```

After calling this method, the EVENT-GROUP-ID attribute is cleared for all connected audit-enabled databases and is no longer recorded in audit event records for this event group.

There can be only one active event group per session at any one point in time. To set a different event group for the session, you can:

- Call the END-EVENT-GROUP( ) method, to end the current event group, and then call the BEGIN-EVENT-GROUP( ) method to begin the new event group.
- Call the BEGIN-EVENT-GROUP( ) method to begin the new event group. If there is an existing event group in effect, the AVM ends the existing event group before beginning the new event group.

Calling this method does not generate an audit event or an audit record.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

See also

BEGIN-EVENT-GROUP( ) method, EVENT-GROUP-ID attribute

END-FILE-DROP( ) method
(Windows only; Graphical interfaces only)

Terminates a drag-and-drop operation and frees the memory allocated by Windows to hold the names of the dropped files.

Return type: BOOLEAN

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

Syntax

```
END-FILE-DROP ( )
```

This method returns TRUE if it is successful. If there is no current drag-and-drop operation, GET-DROPPED-FILE( ) returns FALSE.

END-USER-PROMPT attribute

A freeform string that WebClient uses when prompting for a userid and password, if it does not find those values in the security cache.

Data type: CHARACTER

Access: Read-only

Applies to: CODEBASE-LOCATOR system handle

ENTITY-EXPANSION-LIMIT attribute

Indicates the maximum number of entity substitutions the XML parser (either DOM or SAX) will permit in a document.

Data type: INTEGER

Access: Readable/Writable

Applies to: SAX-reader object handle, X-document object handle

By limiting the number of entity substitutions, use of this attribute prevents recursive entities from consuming system resources. The default value is 50,000. The attribute's value must be equal to or greater than 0; setting the attribute to a negative value will result in a runtime error. If the attribute is set to the Unknown value (?), the feature is off.

You can set this behavior globally by using the `-entityExpansionLimit` startup parameter. The default value of the attribute is set by the startup parameter, but you can change the attribute as needed to override the behavior set by the startup parameter. See the *Startup Command and Parameter Reference* for more information.

ENTRY( ) method

Returns the character-string value of the specified list entry.

Return type: CHARACTER

Applies to: BROWSE widget (column), COMBO-BOX widget, SELECTION-LIST widget

Syntax

```
ENTRY ( list-index )
```

list-index

An integer expression that specifies an entry within the combo-box list or selection list.

For browses, this method applies only to combo-box browse columns.

ENTRY-TYPES-LIST attribute

Returns a character string containing a comma-separated list of all valid entry types for the current OpenEdge environment. For DataServer server logging, the string contains the valid entry types against the logical database specified by the DB-CONTEXT attribute.

Data type: CHARACTER

Access: Read-only

Applies to: DSLOG-MANAGER system handle, LOG-MANAGER system handle

ERROR attribute

A compile-time or run-time error condition.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Asynchronous request object handle, Buffer object handle, COMPILER system handle, ERROR-STATUS system handle, ProDataSet object handle, Temp-table object handle

The ERROR attribute corresponds to the ERROR function.

For the asynchronous request object handle, the ERROR attribute indicates that the ERROR condition was encountered during the processing of an application server or Web Server request. If the COMPLETE attribute is FALSE, the value of this attribute is the Unknown value (?). This attribute is set immediately before the associated event procedure or method is executed.

For the COMPILER system handle, the ERROR attribute indicates whether an error occurred in the preceding compilation. If no error occurred in the preceding compilation, the value of ERROR is the Unknown value (?).

For the ProDataSet object handle, Temp-table object handle, and Buffer object handle, the ERROR attribute indicates whether an error occurred during a FILL or SAVE-ROW-CHANGES operation.

For a ProDataSet object handle or a temp-table object handle, the ERROR attribute is reset to FALSE when one of the following methods removes the object's before-image records:

- FILL( ) method
- EMPTY-DATASET( ) method
- MERGE-CHANGES( ) method
- ACCEPT-CHANGES( ) method
- REJECT-CHANGES( ) method

Note:

The ERROR attribute is effective only if a before-table is NO-UNDO during a data source update. If the before-table is UNDO, the ERROR attribute is lost.

For the Buffer object handle, this attribute causes the row to be backed out rather than merged with the MERGE-CHANGES( ) method or the MERGE-ROW-CHANGES( ) method.

This attribute is marshalled between the client and the application server.

For the ERROR-STATUS system handle, the ERROR attribute indicates whether the ABL ERROR condition was raised in the most recent statement that used the NO-ERROR option. If no ERROR condition was raised in that statement, the value of the ERROR attribute is FALSE.

The ERROR attribute is writeable for a ProDataSet object, a Temp-table object, and a Buffer object. You can set the ERROR attribute programmatically in order to manage the application response to errors. For example, you might set the ERROR attribute to FALSE in order to ignore all errors that might be returned for a statement using the NO-ERROR option. However, note that manually setting this attribute has no affect on ABL error management. Thus, if you set the ERROR attribute to TRUE, the AVM does **not** also raise the ERROR condition.

Note: Statements such as COMPILE that handle errors as part of their normal function do not raise the ABL ERROR condition. For example, the COMPILE statement does not raise the ERROR condition when it encounters compilation errors in a procedure.

ERROR-COLUMN attribute

The character position at which a compiler error occurred.

Data type: INTEGER

Access: Read-only

Applies to: COMPILER system handle

If no error occurred in the preceding compilation, the value of ERROR-COLUMN is the Unknown value (?).

ERROR-OBJECT attribute

An object reference to an instance of a class that implements the `Progress.Lang.Error` interface. The instance describes an ERROR condition raised by an asynchronous call to a procedure or Web service operation. This attribute holds the Unknown value (?) if there is no ERROR or relevant STOP condition.

Note: If the object reference is to an instance of `Progress.Lang.StopError`, it describes a STOP condition raised from an unexpected error in the asynchronous call, not an ERROR condition itself.

Data type: Class that implements the Progress.Lang.Error interface

Access: Read-only

Applies to: Asynchronous request object handle

The error or stop object the attribute references can be thrown by the server or raised in the client before anything is run on the server, for example, if there is an error in processing one of the parameters.

Compared to the ERROR-STATUS system handle , ERROR-OBJECT can accommodate user-defined error objects, thereby providing more information to the client.

Notes

- Error objects can be thrown by the server:
  - From an error CATCH block
  - Using the ROUTINE-LEVEL ON ERROR UNDO, THROW directive

    - Using a `Progress.Lang.AppError` object that is created as a result of a RETURN ERROR in the called procedure

- `StopError` objects can be thrown by the server:
    - From a stop CATCH block
    - Whenever the server does not handle the corresponding STOP condition and the client has a relevant CATCH block on its portion of the call stack

- If a `StopError` object is thrown from the server, its error messages continue to be available on the client from the serialized object even though they are written to the log file on the OpenEdge application server.

See also

STOP-OBJECT attribute, Progress.Lang.Error interface, Progress.Lang.StopError class

ERROR-OBJECT-DETAIL attribute

Identifies the SOAP-fault object that contains SOAP fault message detail.

Data type: HANDLE

Access: Read-only

Applies to: ERROR-STATUS system handle

If a Web service operation generates a SOAP fault message, the AVM generates the following error:

Syntax

```
Web service %s<operation> failed.  SOAP faultstring is %s (nnnn)
```

The complete SOAP fault error message is returned to the AVM as part of the ERROR-STATUS system handle.

If the ABL application invokes the Web service operation with the NO-ERROR option on the RUN statement, any errors that occur as a result of the operation are suppressed. In this case, the application can access the SOAP fault message detail using the SOAP-fault and SOAP-FAULT-DETAIL object handles. Otherwise, the AVM displays the error message to the end user.

ERROR-ROW attribute

The line number at which a compiler error occurred.

Data type: INTEGER

Access: Read-only

Applies to: COMPILER system handle

If no error occurred in the preceding compilation, the value of ERROR-ROW is the Unknown value (?).

ERROR-STACK-TRACE attribute

This attribute controls whether or not the `CallStack` property on error objects can be used. Because the `CallStack` feature is for debugging and can slow system performance, error objects should not save call stacks in a production environment. This attribute allows you to disable the `CallStack` property on all error objects. When this attribute is FALSE, the default value, the `CallStack` property on an error object will be the Unknown value (?).

You can also set this session attribute using the `-errorstack` startup parameter.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

An error object thrown from an application server to a client will contain the stack trace from the application server in its `CallStack` property. If ERROR-STACK-TRACE is TRUE on the client, the AVM will prepend the `CallStack` property with the stack trace from the client. The text "Server StackTrace" will appear at the start of the application server stack trace to differentiate it from the client side stack. In the example below, the lines in boldface represent the application server stack trace:

```
doIt asclient.p at line 46  (asclient.p)
asclient.p at line 26  (asclient.p)
main.p at line 10  (main.p)

Server StackTrace:
myproc svr.p at line 30 (svr.p)
svr.p at line 5  (svr.p)
```

ERROR-STRING attribute

An arbitrary string value associated with a buffer or temp-table object that provides descriptive information about an error on that object.

Data type: CHARACTER

Access: Readable/Writable

Applies to: Buffer object handle, Temp-table object handle

Setting the attribute does not in any way signal an error condition to ABL. The AVM does not inspect this string, or take action based on its value.

The AVM automatically clears this attribute by setting its value to the empty string ("") when one of the following occurs:

- The FILL( ) method is used on any ProDataSet object containing a temp-table, on one of its member temp-table objects, or on a parent buffer that cascades down through that temp-table object.
- The EMPTY-TEMP-TABLE( ) method is used on a temp-table object buffer.
- You set the ERROR-STRING attribute, for the temp-table object, to the empty string ("").

The number of characters in this string is limited to 3K.

Note: The ERROR-STRING attribute is effective only if a before-table is NO-UNDO during a data source update. If the before-table is UNDO, the ERROR-STRING attribute is lost.

EVENT-GROUP-ID attribute

Returns the universally unique identifier (UUID) for the audit event group in effect for the current session, as a Base64 character string. The UUID is 22 characters in length (the two trailing Base64 pad characters are removed).

Data type: CHARACTER

Access: Read-only

Applies to: AUDIT-CONTROL system handle

The value of this attribute is set by the BEGIN-EVENT-GROUP( ) method, and cleared by the END-EVENT-GROUP( ) method. This value is recorded in all audit event records generated for this audit event group until you either end the current audit event group or begin a different audit event group.

If no audit event group is in effect, this method returns the Unknown value (?).

See also

BEGIN-EVENT-GROUP( ) method, END-EVENT-GROUP( ) method

EVENT-HANDLER attribute

The name of the internal procedure or method to run on completion of an ABL event for an asynchronous request.

Data type: CHARACTER

Access: Read-only

Applies to: Asynchronous request object handle

EVENT-HANDLER-OBJECT attribute

For an asynchronous object, returns the object specified by `EVENT-HANDLER-CONTEXT` in the `RUN` statement. Otherwise, returns the Unknown value (**?**).

For the `SAX-reader` object, it is set to an object-oriented ABL object reference that is the callback handler for the `SAX-reader` callback events.

Data type: `Progress.Lang.Object`

Access: Read-only for the Asynchronous request object handle, Readable/Writable for the SAX-reader object handle

Applies to: Asynchronous request object handle, SAX-reader object handle

Notes: The EVENT-HANDLER-OBJECT attribute shows the following behavior when used with the SAX-reader object:

- Assigning a value other than (**?**) to the EVENT-HANDLER-OBJECT attribute will cause an update to the HANDLER attribute changing it to the unknown value.
- Assigning a value other than (**?**) to the HANDLER attribute will cause an update to the EVENT-HANDLER-OBJECT attribute changing it to the unknown value.
- Assigning a value of unknown to the EVENT-HANDLER-OBJECT attribute will unregister any existing object-oriented ABL object callback context.
- The assigned value must be a handle and may be any of the following:
  - THIS-OBJECT, when called from within a method.
  - An expression that results in an object handle variable that refers to a valid object-oriented ABL object instance.

EVENT-PROCEDURE attribute

The name of the internal procedure to run as the event procedure for an asynchronous request.

Note: Applies only if the ASYNCHRONOUS attribute is TRUE.

Data type: CHARACTER

Access: Read-only

Applies to: Asynchronous request object handle, Call object handle

The name of this internal procedure is the same as the name of the event procedure as specified by the EVENT-PROCEDURE option on the RUN statement. If the EVENT-PROCEDURE option is not specified, this attribute is set to the empty string (""). The attribute returns the unknown (?) value if the EVENT-HANDLER-OBJECT attribute is specified as an object.

EVENT-PROCEDURE-CONTEXT attribute

The procedure handle of the active procedure context where the event procedure is defined for an asynchronous request.

Note: Applies only if the ASYNCHRONOUS attribute is TRUE.

Data type: HANDLE

Access: Read-only

Applies to: Asynchronous request object handle, Call object handle

This procedure handle is the same as the handle specified by the EVENT-PROCEDURE...IN *procedure-context* option of the RUN statement that executes this request. If the EVENT-PROCEDURE...IN option is not specified (the default), this attribute is set to the value of the THIS-PROCEDURE system handle at the time the RUN statement was executed. The attribute returns the unknown (?) value if the EVENT-HANDLER-OBJECT attribute is specified as an object.

EVENT-TYPE attribute

The type of the last event.

Data type: CHARACTER

Access: Read-only

Applies to: LAST-EVENT system handle

Valid event types are:

- **"KEYPRESS"** — When the detected event is a keyboard event identified by key label, such as ESC, CTRL+A, or A.
- **"MOUSE"** — When the detected event is a portable or three-button mouse event, such as MOUSE-SELECT-UP or LEFT-MOUSE-UP.
- **"PROGRESS"** — When the detected event is a high-level ABL event. These include all events identified as direct manipulation, key function, developer, and other miscellaneous events, such as SELECTION, DELETE-LINE, U1, or CHOOSE.

EXCLUSIVE-ID attribute

The ID assigned to a state-aware cookie. Intended for internal use only.

Data type: CHARACTER

Access: Read-only

Applies to: WEB-CONTEXT system handle

EXECUTION-LOG attribute

Determines whether the AVM traces the execution of procedures, user-define functions, and the publish and subscribe statements in the execution log. Intended for internal use only.

Data type: LOGICAL

Access: Readable/Writable

Applies to: SESSION system handle

EXIT-CODE attribute

Returns an optional exit code set by the application, when the AVM process ends.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SESSION system handle

The attribute allows the application to optionally set a program exit code. This provides meaningful feedback to the shell script and allows the creation of automation scripting around the termination of the ABL application.

This attribute is set to the Unknown value (?) if not set by the application.

For example:

```
SESSION:EXIT-CODE=123.


SESSION:EXIT-CODE=myIntVar.
```

Windows allows positive integer values (0-32,767). UNIX restricts this to the range of 0 through 255 (inclusive). If you attempt to set an improper value for your platform, a runtime error is raised.

If an AVM process running as an application server session sets an `EXIT-CODE` value, a runtime error is raised.

The exit code specified by `SESSION:EXIT-CODE` can be printed from a bash shell as shown in the following example:

```
$PROEXE -p MyTest.p -b


echo $?
```

In Windows, the exit code can be printed from the command line as shown below:

```
start /wait $PROEXE -p MyTest.p -b


echo %errorlevel%
```

In Windows PowerShell the pseudo-environment variable is called by `$LastExitCode` instead of `%errorlevel%`.

Other shells in Windows may restrict the exit code to a range smaller than 0-32,767. For example, Cygwin limits the value to the same range as Unix (0-255) and returns only the lower 8 bits of the value. For example, if the application sets the exit code to 300, the exit code displayed from a Windows command prompt (`cmd.exe`) is 300, while in Cygwin it is 44 (the lower 8 bits):

Integer value	Binary value
300	00000000 00000000 00000001 00101100
44	00000000 00000000 00000000 00101100

It is important to keep this in mind when choosing exit code values.

EXPAND attribute

How to set the size of a horizontal radio set.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: RADIO-SET widget

This attribute applies to radio sets whose HORIZONTAL attribute are set to TRUE (for horizontal alignment).

If TRUE, the size for each button is equal to the width of the button with the longest label. If FALSE, the size for each button is set according to its label.

You can set this attribute only before the widget is realized.

EXPANDABLE attribute
(Graphical interfaces only)

Indicates whether the AVM extends the right-most browse column to the right edge of the browse. This covers white space that appears when the browse is wider than the sum of the widths of the browse columns.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget

If you set a browse's EXPANDABLE attribute to TRUE, the AVM extends the right-most browse column to the right edge of the browse, if necessary, to cover any white space that might appear-unless you explicitly set the width of the right-most browse column using the WIDTH-CHARS or WIDTH-PIXELS attribute. The right-most browse column expands to the right anytime the browse or another browse column is resized.

If the browse has a horizontal scroll bar, no white space appears between the right-most browse column and the right edge of the browse, and the right-most browse column does not expand to the right.

If EXPANDABLE is TRUE and a browse column's VISIBLE attribute is changed, the last column's width may be changed.

When adding dynamic browse columns to a browse, it is best to keep EXPANDABLE turned off until all columns are added.

EXPORT( ) method
(PAS for OE only)

Creates and modifies an application server's export list, which specifies the remote procedures that a client application can execute in the current application server session.

Return type: LOGICAL

Applies to: SESSION system handle

Syntax

```
EXPORT ( [ list ] )
```

list

A comma-separated list of procedure names and name-patterns. EXPORT( ) ignores white space (blank, tab, and newline) at the beginning and end of an entry. The only wildcard EXPORT( ) supports

is the asterisk (*). For more information on wildcards, see the reference entry for the MATCHES function.

The EXPORT( ) method applies only to application servers. That is, if the REMOTE attribute of the SESSION handle is FALSE, EXPORT( ) does nothing and returns FALSE.

The EXPORT( ) method can be called repeatedly within the context of the application server instance. Each time EXPORT( ) is called, the application server instance adds the procedures in *list* to its export list. If you do not specify *list*, the EXPORT( ) method resets the export list to empty.

The EXPORT( ) method performs pattern matching by comparing two procedure names character-by-character, taking wildcards into account. Procedure names must match exactly. Case (uppercase and lowercase) is significant.

If the EXPORT( ) method is never called, a client application can call any procedure in the application server's PROPATH. Once EXPORT( ) is called in the context of an application server, a client application can call only the procedures in the export list.

Typically, the Connect procedure or Startup procedure of an application server calls the EXPORT( ) method, depending on the operating mode. For example, where you might call it from the Connect procedure on a session-managed application server, you would probably call it from the Startup procedure on a session-free application server. For more information on Connect procedures, see the reference entry for the CONNECT( ) method (Progress Application Server for OpenEdge).

EXPORT-PRINCIPAL( ) method

Exports the unsealed or sealed client-principal object as a portable security token with the currently defined property and attribute settings by converting it to a RAW value. You can assign this value to a RAW variable and send it to another ABL session (such as an application server session) or assign it to a field in a secure database. The receiving ABL session can then import the RAW value into another client-principal object using the IMPORT-PRINCIPAL( ) method. Depending on its sealed or unsealed state, you can use the imported object to assert or set the user identity it represents using the SEAL( ) method, SECURITY-POLICY:SET-CLIENT( ) method, or SET-DB-CLIENT function, as appropriate.

Return type: RAW

Applies to: Client-principal object handle

Syntax

```
EXPORT-PRINCIPAL( )
```

The client-principal object can be either sealed or unsealed.

Calling this method does not generate an audit event or an audit record.

Notes

- A typical use case for exporting an unsealed security token is having a client log in by exporting an unsealed client-principal initialized with its asserted user identity to a remote authentication service over a secure connection. The authentication service then imports the client-principal and, after a successful user authentication, seals the client-principal, beginning a user login session. The service might then export and return the sealed client-principal to the requesting client, which imports the object to set its authenticated identity. The service might also cache the exported client-principal in secure storage for a future identity management request.

- A typical use case for exporting a sealed security token is in a remote authentication service that responds to identity management requests from ABL sessions of an n-tier application. For each request, the authentication service retrieves and imports the sealed client-principal from secure storage that is associated with a given login session key and performs the requested action (such as invoking the LOGOUT( ) method to terminate the user login session). The service then exports the sealed (and changed) client-principal, replacing the previous copy in secure storage and possibly returning it to the requesting ABL session.

See also

IMPORT-PRINCIPAL( ) method, SET-CLIENT( ) method, SET-DB-CLIENT function

EXTENT attribute

The number of elements in an array field.

Data type: INTEGER

Access: Read-only

Applies to: Buffer-field object handle

Note: The EXTENT attribute corresponds to the EXTENT function.

FETCH-SELECTED-ROW( ) method

Fetches the *n*th selected row in a browse and puts the row into the database buffer. In other words, this method specifies one row from the one-based index into all currently selected rows and puts that row into the record buffer.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
FETCH-SELECTED-ROW ( n )
```

n

An integer expression that specifies a selected row within the browse.

The AVM maintains a numbered list of selected rows, starting at 1. The AVM builds this numbered list as the user selects rows in the browse. When you call the FETCH-SELECTED-ROW method, the AVM searches this list to find the *n*th row selected by the user.

FGCOLOR attribute
(Graphical interfaces only)

The color number for the foreground color of the widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse, column, and cell), COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

For a browse cell, it specifies the color of a specific cell in the view port. You can set this color only as the cell appears in the view port during a ROW-DISPLAY event.

You cannot set this attribute for a BUTTON widget.

You cannot set this attribute for combo-boxes, selection-lists, editors, or native fill-in fields (VIEW-AS FILL-IN NATIVE) when they are disabled (not SENSITIVE).

Setting the FGCOLOR attribute for any of the following field-level widgets within a dialog-box or frame overrides any INHERIT-FGCOLOR attribute or option settings: BROWSE, COMBO-BOX (all types), EDITOR, FILL-IN (NATIVE and Enabled), and SELECTION-LIST.

Note: This attribute has no meaning for control-frames because the ActiveX control visualization constitutes the foreground.

See also

BGCOLOR attribute, Frame phrase, INHERIT-FGCOLOR attribute

FILE-CREATE-DATE attribute
(Windows only)

Indicates the date on which the specified file was created.

Data type: DATE

Access: Read-only

Applies to: FILE-INFO system handle

FILE-CREATE-TIME attribute
(Windows only)

Indicates the time when the specified file was created.

Data type: INTEGER

Access: Read-only

Applies to: FILE-INFO system handle

FILE-MOD-DATE attribute

Indicates the last date the specified file was modified.

Data type: DATE

Access: Read-only

Applies to: FILE-INFO system handle

This attribute is supported on all platforms.

FILE-MOD-TIME attribute

Indicates the last time the specified file was modified.

Data type: INTEGER

Access: Read-only

Applies to: FILE-INFO system handle

This attribute is supported on all platforms.

FILE-NAME attribute

The name of the file associated with a handle. Returns the empty string for a Web service procedure.

The FILE-NAME attribute of the COMPILER handle is maintained only for backward compatibility. Use the GET-FILE-NAME( ) method, instead.

Data type: CHARACTER

Access: Readable/Writable

Applies to: COMPILER system handle, FILE-INFO system handle, Procedure object handle, PROFILER system handle, RCODE-INFO handle, SOURCE-PROCEDURE system handle, TARGET-PROCEDURE system handle, THIS-PROCEDURE system handle

The FILE-NAME attribute of the COMPILER handle is the name of the source file from the preceding compilation. If no error occurred during the preceding compilation, FILE-NAME assumes the unknown value (?).

For the PROFILER handle, FILE-NAME is the file into which the profiling data is written after the raw recorded data is analyzed. The file name must be set before data can be stored. The default value is the unknown value (?).

The FILE-NAME attribute of the FILE-INFO or RCODE-INFO handle is the name of the file used by subsequent references to the handle. You can specify the filename with a `.p`, `.r`, or no extension. If you set FILE-NAME to a relative pathname, then the AVM searches the `PROPATH` to find the file. Otherwise, the AVM looks for the file specified by the absolute pathname. When used with FILE-INFO, the filename can contain Unicode characters. When used with RCODE-INFO, the filename cannot contain characters outside of the non-Unicode code page. For more information about Unicode and code pages, see *Internationalize ABL Applications*.

The FILE-NAME attribute of a procedure handle is the pathname of the procedure file that contains the procedure associated with the handle. If the procedure file is specified by the Startup Procedure (`-p`) parameter, then the attribute contains the full pathname of the file. Otherwise, it contains the pathname exactly as specified in the RUN statement that invoked it. The procedure can be local or remote.

The FILE-NAME attribute of the COMPILER handle and procedure handle is read only.

FILE-OFFSET attribute

The character offset in the source file in which a Compiler error occurred.

Data type: INTEGER

Access: Read-only

Applies to: COMPILER system handle

If no error occurred in the preceding compilation, the value of FILE-OFFSET is the Unknown value (?).

FILE-SIZE attribute

Indicates the size of the specified file.

Data type: INTEGER

Access: Read-only

Applies to: FILE-INFO system handle

This attribute is supported on all platforms.

FILE-TYPE attribute

A string of characters that indicate the type of file that is currently specified for the FILE-INFO handle.

Data type: CHARACTER

Access: Read-only

Applies to: FILE-INFO system handle

The character string specifies two classes of file types—one type per file from the first class and one or more types per file from the second class. The following table lists the file type characters from the first class of file types.

Table 78: File type characters—one per file

File type	Description
D	The file is a directory.
F	The file is a standard file or FIFO pipe (UNIX systems).
M	The file is a member of an ABL procedure library.
S	The file is a special device (UNIX systems).
X	The file type is unknown. (Contact your Progress Software Corporation Technical Support representative if you receive this value.)

The following table lists the file type characters from the second class of file types.

Table 79: File type characters—one or more per file

File type	Description
H	The file is hidden.
L	The file is a symbolic link (UNIX systems).
P	The file is a pipe file (UNIX systems).
R	The file is readable.
W	The file is writeable.

FILL( ) method

Fills a ProDataSet object, recursively, based on its defined data sources, data relations, and queries. You can fill a ProDataSet object completely by running FILL on the ProDataSet handle itself, or partially by starting at the level of one of its member Buffer objects.

Return type: LOGICAL

Applies to: ProDataSet object handle, Buffer object handle

Syntax

```
FILL( )
```

You can define a query for the data source of a ProDataSet member buffer at any level of the ProDataSet object to select the records to fill in one FILL operation. You can also use the FILL-WHERE-STRING attribute to override the WHERE clause in the query for the data source during a FILL operation.

You can perform a FILL operation on a ProDataSet object or one of its member buffer objects any number of times. You might do this, for example, to load data in a ProDataSet object or buffer after you have modified a data-source object query or attached to a different data-source object.

You can specify the FILL mode to direct the FILL operation for a ProDataSet member buffer using the FILL-MODE attribute. The default FILL-MODE is MERGE. When FILL-MODE is REPLACE, the BEFORE-ROW-FILL event handler is run after the original ProDataSet row is copied into the ProDataSet buffer.

A number of built-in events fire automatically at different points during the FILL process, for example before the FILL, after a record is created in the temp-table, and so on. For more information about ProDataSet events, see the ProDataSet events.

You can recursively fill a ProDataSet with self-referencing elements, for example, bill-of-materials parent/child tables or organization chart manager/employee tables. For a static ProDataSet, use the RECURSIVE option of the data-relation object. For a dynamic ProDataSet, use the ADD-PARENT-ID-RELATION( ) method of the ProDataSet object handle.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

If the AVM encounters an error, it sets the value of the ERROR attribute to TRUE for the associated ProDataSet object handle and Temp-table object handle.

Note: When executing the method, there is no guarantee as to the contents of the buffers involved after the method finishes executing.

For more information about filling a ProDataSet object, see *Use ProDataSets*.

See also

ADD-PARENT-ID-RELATION( ) method, ATTACH-DATA-SOURCE( ) method, BATCH-SIZE attribute, CREATE DATASET statement, CREATE DATA-SOURCE statement, DEFINE DATASET statement, DEFINE DATA-SOURCE statement, FILL events, FILL-MODE attribute, FILL-WHERE-STRING attribute, LAST-BATCH attribute, NEXT-ROWID attribute, RECURSIVE attribute, RESTART-ROWID attribute, SET-CALLBACK-PROCEDURE( ) method

FILLED attribute

Indicates if the background color of a rectangle is set to a certain value.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: RECTANGLE widget

If the value of the FILLED attribute is TRUE, the background color of the rectangle depends on the value of the BGCOLOR attribute (for graphical interfaces) or the value of the DCOLOR attribute (for character interfaces). The default value of FILLED is TRUE.

FILL-MODE attribute

Specifies the mode in which the FILL( ) method fills a ProDataSet member buffer. The default mode is MERGE.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer object handle

The following table lists the FILL( ) method modes.

Table 80: FILL( ) method modes

When the mode is . . .	The FILL( ) method . . .
APPEND	Fills the table by adding new records on top of existing records in the table, without performing any record comparisons. If this creates duplicate records, the AVM generates a run-time error and you must manage the duplicate records. If you are certain there are no duplicate records, an APPEND is more efficient than a MERGE.
EMPTY	Empties the table before the FILL operation begins.

When the mode is . . .	The FILL( ) method . . .
MERGE	Fills the table by merging new records with existing records in the table. The FILL( ) method checks each record to ensure there are no duplicate records (based on the table's unique primary index). If the FILL( ) method finds a record with a duplicate key, it does not replace the record because the record might have dependent records elsewhere in the ProDataSet. In this case, the AVM does not generate a run-time error. Thus, you cannot use this mode to refresh existing records.
NO-FILL	Does not perform the FILL operation on the table.
REPLACE	Fills the table by replacing existing records in the table. The FILL( ) method checks each record to determine whether or not it exists in the table (based on the table's unique primary index). If the record exists in the table, the FILL( ) method replaces it. If the record does not exist, the FILL( ) method creates a new record. The BEFORE-ROW-FILL event handler is run after the original ProDataSet row is copied into the ProDataSet buffer. A REPLACE is less efficient than an APPEND or a MERGE.

FILL-WHERE-STRING attribute

The current WHERE expression for a data-source query, beginning with the keyword WHERE, and not including the prior FOR EACH phrase of an OPEN QUERY statement.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Data-source object handle

You can use this attribute to override the existing WHERE expression in the query for the data-source object during a FILL( ) operation. The default is the null value.

Note:

You can also modify the query on an associated data-relation using the WHERE-STRING attribute.

The NAME attribute of the Temp-table object handle is writeable for dynamic and AVM-generated temp-tables. You might need to update a FILL-WHERE-STRING that references a renamed temp-table with new strings using the new table name.

See also

DEFINE DATA-SOURCE statement, FILL( ) method, WHERE-STRING attribute

FIND-BY-ROWID( ) method

Locates the record with the rowid you specify, then moves the record into the buffer.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
FIND-BY-ROWID ( rowid
  [ ,
    { SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK }
    [ , NO-WAIT ]
  ] )
```

rowid

An expression of type ROWID that represents the rowid of the desired record.

SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK

The type of lock that the AVM places on the record, if found. The default is SHARE-LOCK.

Note: For more information on record locks, see *OpenEdge Getting Started: ABL Essentials*.

NO-WAIT

Causes FIND-BY-ROWID to return FALSE immediately if another user has a lock on the desired record and FIND-BY-ROWID specifies a locking option other than NO-LOCK.

Note: To determine whether another user has a lock on the desired record, use the LOCKED attribute of the buffer object.

The FIND-BY-ROWID method returns TRUE if it finds the record, and FALSE if it does not.

The following is an example:

```
DEFINE VARIABLE bh AS HANDLE NO-UNDO.
DEFINE VARIABLE r  AS ROWID  NO-UNDO.

r = ... /* ROWID from some parameter */
bh = BUFFER CUSTOMER:HANDLE.
bh:FIND-BY-ROWID(r).
```

Note: The FIND-BY-ROWID method corresponds to a FIND statement of the form FIND *buffer* WHERE ROWID (*buffer*) = *rowid* **...**, etc. That is, triggers are honored, and the default lock mode is SHARE-LOCK. One difference, however, is that the FIND-BY-ROWID method does not raise an error if it cannot find the record.

FIND-CURRENT( ) method

Changes the lock mode of a record in a buffer.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
FIND-CURRENT ( [lockmode[ , waitmode]] )
```

lockmode

An integer expression evaluating to one of the following constants: SHARE-LOCK, EXCLUSIVE-LOCK, or NO-LOCK. You can assign any of these constants to an integer variable. For example, mylock = NO-LOCK. The default is SHARE-LOCK.

waitmode

An integer expression evaluating to one of the following: NO-WAIT, 0, or the Unknown value (?). You can assign NO-WAIT to an integer variable. For example, mywait = NO-WAIT.

The default is to wait.

The following shows an example of the FIND-CURRENT method:

```
DEFINE VARIABLE bh AS HANDLE NO-UNDO.

bh = BUFFER Customer:HANDLE.

DO TRANSACTION:
   bh:FIND-CURRENT(EXCLUSIVE-LOCK).
END.
 MESSAGE bh:CURRENT-CHANGED Customer.CustNum.
```

If the change in lock status succeeds, the method returns TRUE, otherwise it returns FALSE.

If the lock change fails, a message displays. You can suppress the message using NO-ERROR on the statement containing the method.

Executing the FIND-CURRENT method resets the CURRENT-CHANGED attribute. If the record in the database changes between the time the original record was found and the FIND-CURRENT executes, the CURRENT-CHANGED attribute returns TRUE. If the record does not change, then the CURRENT-CHANGED attribute returns FALSE.

See also

FIND-BY-ROWID( ) method, CURRENT-CHANGED attribute, CURRENT-CHANGED function, FIND-FIRST( ) method, FIND-LAST( ) method, FIND-UNIQUE( ) method, FIND statement

FIND-FIRST( ) method

Gets a single record. This method lets a user get the first record that satisfies the predicate expression.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
FIND-FIRST ( predicate-expression [ , lockmode[ , waitmode]] )
```

predicate-expression

A character expression that evaluates to the following syntax:

```
[ WHERE [ logical-expression ]][ USE-INDEX index-name]
```

Once evaluated, *predicate-expression* can contain only constants and unabbreviated references to fields from the buffer.

The *predicate-expression* itself can be built using a concatenation of character expressions.

lockmode

An integer expression evaluating to one of the following constants: SHARE-LOCK, EXCLUSIVE-LOCK, or NO-LOCK. You can assign any of these constants to an integer variable. For example, mylock = NO-LOCK.

The default is SHARE-LOCK.

waitmode

An integer expression evaluating to one of the following: NO-WAIT, 0, or the Unknown value (?). You can assign NO-WAIT to an integer variable. For example, mywait = NO-WAIT. The default is to wait.

The following shows some examples of FIND-FIRST method:

```
DEFINE VARIABLE bh     AS HANDLE    NO-UNDO.
DEFINE VARIABLE myname AS CHARACTER NO-UNDO.

bh = BUFFER Customer:HANDLE.

bh:FIND-FIRST("WHERE Customer.CustNum > 2", NO-LOCK).

DO TRANSACTION:
  bh:FIND-FIRST("", EXCLUSIVE-LOCK).
END.

bh:FIND-FIRST("WHERE Customer.Name = " + QUOTER(myname) , NO-LOCK).
```

If FIND-FIRST succeeds, it returns TRUE, otherwise it returns FALSE.

If FIND-FIRST fails, it does not raise an error but displays a message. You can suppress this message by using NO-ERROR on the statement containing the method.

See also

FIND-BY-ROWID( ) method, FIND-CURRENT( ) method, FIND-LAST( ) method, FIND-UNIQUE( ) method, FIND statement

FIND-LAST( ) method

Gets a single record. This method lets a user get the last record that satisfies the predicate expression.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
FIND-LAST ( predicate-expression [ , lockmode [ , waitmode ] ] )
```

predicate-expression

A character expression that evaluates to the following syntax:

```
[ WHERE [ logical-expression ] ] [ USE-INDEX index-name ]
```

Once evaluated, *predicate-expression* can contain only constants and unabbreviated references to fields from the buffer.

The *predicate-expression* itself can be built using a concatenation of character expressions.

lockmode

An integer expression evaluating to one of the following constants: SHARE-LOCK, EXCLUSIVE-LOCK, or NO-LOCK. You can assign any of these constants to an integer variable. For example, mylock = NO-LOCK.

The default is SHARE-LOCK.

waitmode

An integer expression evaluating to one of the following: NO-WAIT, 0, or the Unknown value (?). You can assign NO-WAIT to an integer variable. For example, mywait = NO-WAIT. The default is to wait.

The following shows some examples of the FIND-LAST method:

```
DEFINE VARIABLE bh AS HANDLE NO-UNDO.

bh = BUFFER Customer:HANDLE.

bh:FIND-LAST("WHERE Customer.Balance > 0 use-index name").

bh:FIND-LAST("WHERE Customer.CustNum > 5 AND Customer.Address < 'z'").
```

If FIND-LAST succeeds, it returns TRUE, otherwise it returns FALSE.

If FIND-LAST fails, it does not raise an error but displays an error message. You can suppress the message using NO-ERROR on the statement containing the method.

See also

FIND-CURRENT( ) method, FIND-FIRST( ) method, FIND-UNIQUE( ) method, FIND statement

---

FIND-UNIQUE( ) method

Gets a single record. This method lets a user get a unique record that satisfies the predicate expression.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
FIND-UNIQUE ( predicate-expression [ , lockmode [ , waitmode ] ] )
```

predicate-expression

A character expression that evaluates to the following syntax:

```
[ WHERE [ logical-expression ] ] [ USE-INDEX index-name ]
```

Once evaluated, *predicate-expression* can contain only constants and unabbreviated references to fields from the buffer. The *predicate-expression* itself can be built using a concatenation of character expressions.

lockmode

An integer expression evaluating to one of the following constants: SHARE-LOCK, EXCLUSIVE-LOCK, or NO-LOCK. You can assign NO-LOCK to an integer variable. For example, mylock = NO-LOCK.

The default is SHARE-LOCK.

waitmode

An integer expression evaluating to one of the following: NO-WAIT, 0, or the Unknown value (?). The default is to wait.

You can assign NO-WAIT to an integer variable. For example, mywait = NO-WAIT.

The following shows some examples of the FIND-UNIQUE method:

```
DEFINE VARIABLE bh     AS HANDLE    NO-UNDO.
DEFINE VARIABLE myname AS CHARACTER NO-UNDO.

bh = BUFFER Customer:HANDLE.

bh:FIND-UNIQUE("WHERE Customer.Custnum < 3 AND
Customer.Name = 'lift line skiing'").

bh:FIND-UNIQUE("WHERE Customer.CustNum = 30").

bh:FIND-UNIQUE("WHERE Customer.Name = " + QUOTER(myname).
MESSAGE bh:AMBIGUOUS.
```

If FIND-UNIQUE succeeds, it returns TRUE, otherwise it returns FALSE.

If FIND-UNIQUE fails, a message displays. You can suppress this message using NO-ERROR on the statement containing the method.

If more than one record satisfies the predicate expression, then the AMBIGUOUS attribute is set to TRUE.

See also

FIND-FIRST( ) method, FIND-CURRENT( ) method, FIND-LAST( ) method, FIND statement

FIRST-ASYNC-REQUEST attribute

Returns the first entry in the list of all current asynchronous request handles for the specified application server or Web service that have been created in the current session.

Data type: HANDLE

Access: Read-only

Applies to: Server object handle

If there are no asynchronous request handles for the specified server, FIRST-ASYNC-REQUEST returns the Unknown value (?).

FIRST-BUFFER attribute

Returns the handle for the first dynamic buffer in the first table containing a dynamic buffer. The table may be either a temp-table or a connected database, in that order. If no dynamic temp-table or database buffers exist in the session, it returns the Unknown value (?).

Note: Only dynamic buffers created with the CREATE BUFFER statement are chained on the SESSION system handle.

Data type: HANDLE

Access: Read-only

Applies to: SESSION system handle

There is no LAST-BUFFER attribute associated with the SESSION handle since the chain is one-directional.

FIRST-CHILD attribute

The handle of the first widget created in the container widget or the current session.

Data type: HANDLE

Access: Read-only

Applies to: DIALOG-BOX widget, FIELD-GROUP widget, FRAME widget, MENU widget, SUB-MENU widget, WINDOW widget, SESSION system handle

You can use the FIRST-CHILD attribute to find the first entry in a list of all frames and dialog boxes in a window, all field groups in a frame or dialog box, all widgets in a field group, all menu items in a menu or submenu, or all windows in an ABL session (SESSION system handle). After finding the first entry, you can find the remaining entries in the list by using each widget's NEXT-SIBLING attribute.

FIRST-COLUMN attribute

A handle to the first column in a browse widget, regardless of the value of its READ-ONLY attribute or its VISIBLE attribute.

Data type: HANDLE

Access: Read-only

Applies to: BROWSE widget

After finding the first column, accessing the NEXT-COLUMN attribute of the current column allows you to walk through the browse columns.

FIRST-DATASET attribute

A handle to the first dynamic ProDataSet object created in the current ABL session.

Data type: HANDLE

Access: Read-only

Applies to: SESSION system handle

After finding the first entry, you can find the remaining entries in the list by using the NEXT-SIBLING attribute for each dynamic ProDataSet object. Use the NEXT-SIBLING attribute to get the next entry in the list of ProDataSet object handles created in the current ABL session.

FIRST-DATA-SOURCE attribute

A handle to the first dynamic data-source object created in the current ABL session.

Data type: HANDLE

Access: Read-only

Applies to: SESSION system handle

Use the NEXT-SIBLING attribute to get the next entry in the chain of dynamic data-source object handles created in the current ABL session.

FIRST-FORM attribute
(Windows only; GUI for .NET only)

The object reference for the first .NET `Progress.Windows.IForm` or ABL window `FormProxy` (`Progress.Windows.FormProxy`) object in the list of all valid OpenEdge form objects created in the current ABL session.

Data type: Progress.Windows.IForm interface

Access: Read-only

Applies to: SESSION system handle

Once you get the first object reference in the list, you can use the NextForm property in the `Progress.Windows.IForm` interface to get the next entry in the list of OpenEdge form object references.

Note: To obtain the first class-based object in the list of all valid ABL and .NET objects created in the current ABL session, use the FIRST-OBJECT attribute.

If you instantiate a `Progress.Windows.IForm` in a non-GUI ABL session, the object does not appear in the list of OpenEdge form objects, but only in the list of valid class-based objects.

See also

LAST-FORM attribute, LAST-OBJECT attribute, FIRST-OBJECT attribute, NextForm property, Progress.Windows.FormProxy class, Progress.Windows.IForm interface

FIRST-OBJECT attribute

The object reference for the first class instance in the list of all valid ABL and .NET class instances created in the current ABL session. If there are no class instances in the current session, this attribute returns the Unknown value (?).

Data type: Progress.Lang.Object

Access: Read-only

Applies to: SESSION system handle

Once you get the first object reference in the list, you can use the NEXT-SIBLING property in the Progress.Lang.Object class to get the next entry in the list of ABL and .NET object references.

There are two points worth noting about this attribute with garbage collection enabled:

- The object references maintained by the FIRST-OBJECT attribute and the NEXT-SIBLING property do not count as references for garbage collection. That is, if a class instance is referenced only on the session object chain, it is available for automatic garbage collection.
- When an ABL object that inherits from a .NET object is on the session object chain, it can have circular references that prevent it from being deleted (as with pure ABL objects). When there are no more ABL references to such an ABL-derived .NET object, the object is removed from the object chain, but may still be instantiated because of references to it from .NET. If the application re-obtains a reference to the object in ABL by getting it from .NET, the reference is again placed on the object chain.

Note: To obtain the first .NET form or ABL FormProxy object in the list of all valid form objects created in the current ABL session, use the FIRST-FORM attribute.

See also

FIRST-FORM attribute, LAST-OBJECT attribute, NEXT-SIBLING property, Progress.Lang.Object class

FIRST-OF( ) method

Returns TRUE if the current iteration of the query predicate FOR EACH . . . BREAK BY. . . is the first iteration for a new break group.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
FIRST-OF ( level )
```

level

An integer expression that indicates a BREAK BY group, where 0 is the entire query, 1 is the first BREAK BY, 2 is the second BREAK BY, and so on.

The following example shows how the FIRST-OF( ) method is used to identify the first record of a break group:

```
DEFINE QUERY qCustomer FOR Customer SCROLLING.

QUERY qCustomer:QUERY-PREPARE("FOR EACH Customer WHERE Customer.CustNum > 50
  BREAK BY Customer.Country BY Customer.Comments").
QUERY qCustomer:QUERY-OPEN.

REPEAT WITH TITLE "Customers Break By Country and Comments":
  GET NEXT qCustomer.
  IF NOT AVAILABLE Customer THEN LEAVE.
  DISPLAY
    Customer.Country FORMAT "x(10)"
    Customer.Comments FORMAT "x(20)"
    QUERY qCustomer:FIRST-OF(0) LABEL "First"
    QUERY qCustomer:LAST-OF(0) LABEL "Last"
    QUERY qCustomer:FIRST-OF(1) LABEL "First Country"
    QUERY qCustomer:LAST-OF(1) LABEL "Last Country"
    QUERY qCustomer:FIRST-OF(2) LABEL "First Company"
    QUERY qCustomer:LAST-OF(2) LABEL "Last Company".
END.
```

See also

LAST-OF( ) method, OPEN QUERY statement, QUERY-PREPARE( ) method

FIRST-PROCEDURE attribute

For an application server, returns the first entry in the list of remote persistent, single-run, and singleton procedures that are active on the connected application server. For Web services, returns the first entry in the list of procedure objects associated with the Web service.

Data type: HANDLE

Access: Read-only

Applies to: Server object handle, SESSION system handle

If the current session has no active persistent procedures or the application server has no active remote persistent, single-run, or singleton procedures, FIRST-PROCEDURE returns the Unknown value (?). To find the next procedure given the first, use the NEXT-SIBLING attribute of the procedure handle.

Note: The PERSISTENT attribute of a remote persistent procedure returned by FIRST-PROCEDURE is TRUE only if the RUN statement that executes the procedure was invoked with the PERSISTENT option. The PERSISTENT attribute is FALSE if the persistent procedure was invoked with the SINGLE-RUN or SINGLETON option of the RUN statement.

For information on creating persistent, single-run, or singleton procedures, see the RUN statement reference entry. To check a handle for validity, use the VALID-HANDLE function.

FIRST-QUERY attribute

A handle to the first dynamic query created in the current ABL session.

Data type: HANDLE

Access: Read-only

Applies to: SESSION system handle

Use the NEXT-SIBLING attribute to get the next entry in the chain of dynamic query handles created in the current ABL session. The chain of dynamic query handles includes all automatically generated queries, such as those created for data-relation objects.

FIRST-SERVER attribute

A handle to the first entry in the list of server handles of the current ABL session. This includes both application server objects and Web service server objects.

Data type: HANDLE

Access: Read-only

Applies to: SESSION system handle

The handle associated with the first entry in the list of all server handles created in the current session. If the current session has no server handles, FIRST-SERVER has the Unknown value (?). For more information on server handles, see the Server object handle reference entry.

FIRST-SERVER-SOCKET attribute

A handle to the first entry in the list of all valid server socket handles created in the current session. If there are no server socket handles in this session, FIRST-SERVER-SOCKET returns the Unknown value (?).

Data type: HANDLE

Access: Read-only

Applies to: SESSION system handle

FIRST-SOCKET attribute

A handle to the first entry in the list of all valid socket handles created in the current session. If there are no socket handles in this session, FIRST-SOCKET returns the Unknown value (?).

Data type: HANDLE

Access: Read-only

Applies to: SESSION system handle

FIRST-TAB-ITEM attribute

The first widget in the tab order of a field group.

Data type: HANDLE

Access: Readable/Writeable

Applies to: FIELD-GROUP widget

When you set this attribute, the assigned widget is moved to the first tab position, preceding the widget that was previously at this position. Other widgets in the field group maintain their same relative tab positions.

To set the attribute, you must assign it the handle of a field-level widget or frame that can receive focus from a TAB event and that is also a child of the field group to which the attribute applies. If the FIRST-TAB-ITEM attribute is not set (that is, is the Unknown value (?)), the default first tab position goes to the widget identified by the FIRST-CHILD attribute of the field group.

For more information on how frames owned by a field group participate in the tab order of that field group, see the FRAME widget reference entry .

Note: Any tab reordering that you do with this attribute can be reset by a subsequent ENABLE statement unless you define the frame that owns the field group with the KEEP-TAB-ORDER option. For more information, see the ENABLE statement and Frame phrase reference entries.

FIT-LAST-COLUMN attribute
(Graphical interfaces only)

Allows the browse to be displayed so that there is no empty space to the right and no horizontal scroll bar by potentially widening or shrinking the last browse column's width.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

When this attribute is specified, and the last browse column can be fully or partially displayed in the browse's viewport, then the last browse column's width is adjusted so that it fits within the viewport with no empty space to its right and no horizontal scroll bar.

If the last browse column is fully contained in the viewport with empty space to its right, it grows so that its right edge is adjacent to the vertical scroll bar.

If the last browse column extends outside the viewport, it shrinks so its right edge is adjacent to the vertical scroll bar and the horizontal scroll bar is not needed.

The default value is FALSE.

Note: The FIT-LAST-COLUMN attribute performs the same function as the EXPANDABLE attribute. Progress Software Corporation recommends that you use the FIT-LAST-COLUMN attribute instead of the EXPANDABLE attribute. This recommendation includes replacing EXPANDABLE with FIT-LAST-COLUMN in your current code.

The following shows the DEFINE BROWSE statement syntax with FIT-LAST-COLUMN specified:

```
DEFINE BROWSE b1 QUERY q1
  DISPLAY Customer.CustNum Customer.Name
  ENABLE Customer.CustNum WITH 3 DOWN WIDTH 40 FIT-LAST-COLUMN.
```

The MIN-COLUMN-WIDTH attribute affects the FIT-LAST-COLUMN attribute. As a result, if FIT-LAST-COLUMN is set to TRUE, the last browse column is resized to fit within the viewport only if its width is no smaller than the minimum width. To specify the minimum size that the last browse column's width can be reduced to, use the MIN-COLUMN-WIDTH-PIXELS or MIN-COLUMN-WIDTH-CHARS attribute. See MIN-COLUMN-WIDTH-PIXELS attribute and MIN-COLUMN-WIDTH-CHARS attribute for more information.

FIT-LAST-COLUMN and NO-EMPTY-SPACE are mutually exclusive. If both are specified in the DEFINE BROWSE statement, the compiler displays an error message. If one attribute is set to TRUE while the other attribute is already TRUE, a warning message displays at run time.

FIT-LAST-COLUMN is primarily intended for use in the initial layout of a static browse. It is most useful when laying out a browse with a specified width when you have only a few browse columns and you want to fully use the available space in your viewport.

If the FIT-LAST-COLUMN attribute is set to TRUE, and, subsequently, any browse column's width is changed or the browse's width is changed, then the last browse column's width might be adjusted so that it fits within the viewport with no empty space and no horizontal scroll bar.

When the last browse column's width is set at run time after the browse is realized, then FIT-LAST-COLUMN is ignored.

If the FIT-LAST-COLUMN attribute is set to FALSE, the last browse column's width remains the same and is never changed by ABL.

The FIT-LAST-COLUMN attribute and the EXPANDABLE attribute have the same behavior. Therefore, if you specify the Expand Browse (`-expandbrow`) startup parameter at startup, the FIT-LAST-COLUMN attribute is set to TRUE for each browse in that session.

See also

DEFINE BROWSE statement

FLAT-BUTTON attribute
(Windows only; Graphical interfaces only)

Indicates whether a button is two-dimensional until the mouse passes over it, at which time, a 3D border appears.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BUTTON widget

The FLAT-BUTTON attribute must be set before the button is realized. The default value is FALSE.

Setting the FLAT-BUTTON attribute to TRUE forces the NO-FOCUS attribute to TRUE because the FLAT-BUTTON attribute only works with the NO-FOCUS attribute. Similarly, setting the NO-FOCUS attribute to FALSE forces the FLAT-BUTTON attribute to FALSE.

The mnemonic key (**ALT** accelerator) for a widget will not work if the NO-FOCUS attribute is TRUE because this removes the widget from the tab order. Also, because the widget is not in the tab order, pressing **TAB** will not change focus from the widget.

FOCUSED-ROW attribute

The 1-based index or position of the focused row in the viewport.

Data type: INTEGER

Access: Read-only

Applies to: BROWSE widget

FOCUSED-ROW-SELECTED attribute

Indicates whether the row that has focus is selected.

Data type: LOGICAL

Access: Read-only

Applies to: BROWSE widget

If the row that has focus is selected, FOCUSED-ROW-SELECTED is TRUE . Otherwise, it is FALSE.

FONT attribute
(Graphical interfaces only)

The number of the font of a widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse, column, and cell), BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, LITERAL widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

The font number represents an entry in the font table maintained by the FONT-TABLE handle.

For a browse cell, it specifies the font of a specific cell in the view port. You can set this font only as the cell appears in the view port during a ROW-DISPLAY event.

For browses, this attribute applies only to fill-in and combo-box browse columns.

Note: When the AUTO-RESIZE attribute is set to TRUE, the AVM resizes the following widgets with run-time changes to the FONT attribute: Buttons, Combo boxes, Editors, Fill-ins, Radio sets, Selection lists, Sliders, Texts, and Toggle boxes.

FOREGROUND attribute

Indicates whether the field group is a foreground or a background field group.

Data type: LOGICAL

Access: Read-only

Applies to: FIELD-GROUP widget

If the FOREGROUND attribute is TRUE, the field group is a foreground (data iteration) group. If FOREGROUND is FALSE, the field group is the background group for the frame.

FOREIGN-KEY-HIDDEN attribute

Specifies whether to hide foreign key fields in the child records of a nested data-relation in a ProDataset.

Data type: LOGICAL

Access: Readable/Writable

Applies to: Data-relation object handle

This setting is only valid if the NESTED attribute is also TRUE.

When working with large ProDataSets, omitting foreign keys in nested child records can yield smaller XML documents, more efficient network transfers, and performance gains with the READ-XML( ) and WRITE-XML( ) methods.

Care must be taken when deciding to use this feature. The READ-XML( ) method automatically populates foreign keys in nested child records with the value in the outer parent record when the foreign key is omitted from the XML document. Unless you are sure that the receiver of the XML document will do the same, you should not use this option in your nested data-relations.

For example, while .NET can read this XML document and populate an ADO .NET DataSet, it will create rows in the child DataTable with a null value for the foreign key field.

See also

NESTED attribute

FORM-INPUT attribute

Returns raw HTTP form input that is less than 32K in size. Do not access this attribute.

Data type: CHARACTER

Access: Read-only

Applies to: WEB-CONTEXT system handle

FORM-LONG-INPUT attribute

Returns raw HTTP form input that is greater than 32K in size. The AVM performs no conversion on the data. Do not access this attribute.

Data type: MEMPTR

Access: Read-only

Applies to: WEB-CONTEXT system handle

FORMAT attribute

The text format of a widget or browse-cell.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget (cell), Buffer-field object handle, COMBO-BOX widget, FILL-IN widget, TEXT widget, TOGGLE-BOX widget

For DROP-DOWN-LIST combo-boxes, if you set this attribute with items in the drop-down list, all items are converted to the new format. This attribute is ignored for SIMPLE and DROP-DOWN combo-boxes.

For combo boxes whose entries consist of label-value pairs, the AVM converts all values to the new format.

For browses, this attribute applies only to fill-in and combo-box browse columns.

For browses in Windows, if you modify the FORMAT attribute of a browse-cell, its format changes, but its size does not.

For buffer-fields, the value of the FORMAT attribute does not affect the ABL user interface anywhere. Rather, it controls the output of the STRING-VALUE attribute, and lets users explicitly format non-ABL user interfaces.

Note: When the AUTO-RESIZE attribute is TRUE, the AVM resizes combo box and fill-in field widgets with run-time changes to the FORMAT attribute.

FORMATTED attribute

Determines the format of XML output from a SAX-writer object.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SAX-writer object handle

The default value is FALSE.

TRUE indicates that the SAX-writer should format the document with additional white space, carriage returns, and line feeds, so that the elements display in a hierarchical manner.

FALSE indicates the SAX-writer should create an optimized document that includes no extra white space.

You can read this attribute at all times, but you can only write to it when the WRITE-STATUS is either SAX-WRITE-IDLE or SAX-WRITE-COMPLETE. That is, you can only change the attribute when the SAX-writer is not writing, otherwise the call fails and generate an error message.

FORWARD-ONLY attribute

Lets you avoid building result-lists for static and dynamic queries.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Query object handle

Set to TRUE to avoid building result-lists for queries. Set to FALSE to build result-lists for queries. The default is FALSE.

When TRUE, you cannot use the GET PREV, GET LAST, REPOSITION, or BROWSE methods or statements with these queries. If you do, the AVM generates an error. You can use the GET-FIRST( ) method and GET FIRST statement only on newly opened queries, and you can use the GET NEXT statement and GET-NEXT( ) method freely.

If you set FORWARD-ONLY to TRUE, and you open a query with preselect or sort, the AVM still builds a result-list in order to resolve the query. You cannot set FORWARD-ONLY while a query is open or being browsed.

Setting FORWARD-ONLY to TRUE can improve the performance of operations on queries.

FRAGMENT attribute

Specifies if the output of a SAX-writer object is a complete document or a fragment.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SAX-writer object handle

The default value is FALSE.

TRUE indicates that the writer should not include the XML declaration or require a root node. This behavior allows the developer to create XML fragments which can be used to create larger documents. For example, one SAX-writer object's document fragment LONGCHAR could be used as the parameter of WRITE-FRAGMENT for another SAX-writer.

FALSE indicates the SAX-writer should create a complete XML document with the XML declaration and root node.

You can read this attribute at all times, but you can only write to it when the WRITE-STATUS is either SAX-WRITE-IDLE or SAX-WRITE-COMPLETE. That is, the attribute can only be changed when the SAX-writer is not writing, otherwise it fails and generates an error message.

FRAME attribute

The handle of the frame that contains the widget.

Data type: HANDLE

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

This attribute is writeable only for static frames and all dynamic widgets. You can set this attribute for a static frame only before the widget is realized.

FRAME-COL attribute

The decimal column position, in character units, of the left edge of the widget relative to the upper left corner of the frame that contains the widget.

Data type: DECIMAL

Access: Read-only

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

FRAME-NAME attribute

The name of the frame that contains the widget.

Data type: CHARACTER

Access: Read-only

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

FRAME-ROW attribute

The decimal row position, in character units, of the top edge of the widget relative to the upper left corner of the frame that contains the widget.

Data type: DECIMAL

Access: Read-only

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

If the parent frame is a down frame with multiple occurrences, the FRAME-ROW attribute regards the original occurrence as the parent, not the current occurrence.

FRAME-SPACING attribute

The number of display units between frames in a window. In graphical interfaces the display units are pixels. In character interfaces the display units are character cells.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SESSION system handle

By default, the value for FRAME-SPACING is the height of one row in the default system font. In character interfaces, this is the character cell height. In graphical interfaces, this is the number of pixels returned by the PIXELS-PER-ROW attribute.

FRAME-X attribute

The location of the left edge of the widget relative to the upper left corner of the frame that contains the widget.

Data type: INTEGER

Access: Read-only

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

In character mode, this attribute returns the widget location in row column units. In graphical interfaces, this attribute returns pixels.

FRAME-Y attribute

The location of the top edge of the widget relative to the upper left corner of the frame that contains the widget.

Data type: INTEGER

Access: Read-only

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

In character mode, this attribute returns the widget location in row column units. In graphical interfaces, this attribute returns pixels.

FREQUENCY attribute

Indicates the incremental display of the TIC-MARKS attribute. It is used exclusively with the TIC-MARKS attribute.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SLIDER widget

For example, if you set FREQUENCY to 5, a tic mark appears in every fifth position along the length of the slider.

FULL-HEIGHT-CHARS attribute

The maximum internal height of the window, in character units.

Data type: DECIMAL

Access: Read-only

Applies to: WINDOW widget

The maximum internal height of a window is the height of the screen display minus the vertical spacing required to display the border, title bar, menu bar, message area, and status area of the window.

The value of this attribute is the Unknown value (?) until the window is realized.

FULL-HEIGHT-PIXELS attribute

The maximum internal height of the window, in pixel units.

Data type: INTEGER

Access: Read-only

Applies to: WINDOW widget

The maximum internal height of a window is the height of the screen display minus the vertical spacing required to display the border, title bar, menu bar, message area, and status area of the window.

The value of this attribute is the Unknown value (?) until the window is realized.

FULL-PATHNAME attribute

The absolute pathname of the file specified in the FILE-NAME attribute.

Data type: CHARACTER

Access: Read-only

Applies to: FILE-INFO system handle

The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

FULL-WIDTH-CHARS attribute

The maximum internal width of the window, in character units.

Data type: DECIMAL

Access: Read-only

Applies to: WINDOW widget

The maximum internal width of a window is the width of the screen display minus the horizontal spacing required to display the border of the window.

The value of this attribute is the Unknown value (?) until the window is realized.

FULL-WIDTH-PIXELS attribute

The maximum internal width of the window, in pixel units.

Data type: INTEGER

Access: Read-only

Applies to: WINDOW widget

The maximum internal width of a window is the width of the screen display minus the horizontal spacing required to display the border of the window.

The value of this attribute is the Unknown value (?) until the window is realized.

FUNCTION attribute

The names of high-level events based on the EVENT-TYPE attribute value.

Data type: CHARACTER

Access: Read-only

Applies to: LAST-EVENT system handle

For EVENT-TYPE = "KEYPRESS", this attribute returns key functions, such as "RETURN". For EVENT-TYPE = "MOUSE", this attribute returns high-level events for both portable and three-button event types, such as "MOUSE-SELECT-CLICK" (portable) or "LEFT-MOUSE-CLICK" (three-button). For EVENT-TYPE = "PROGRESS", this attribute returns high-level widget and direct manipulation events, such as "CHOOSE" or "SELECTION".

GET-ATTRIBUTE( ) method

Returns the value of the specified attribute of an element referred to by an XML node reference.

Return type: CHARACTER

Applies to: X-noderef object handle

Syntax

```
GET-ATTRIBUTE ( name )
```

name

The attribute name whose value is desired. Attribute names are defined within the element tag. If using a DTD, you can define attributes with the "IMPLIED" property and those attributes will appear in the DOM structure.

If the requested attribute does not exist, an empty string is returned.

The following example demonstrates listing all the attributes of the node:

```
...
anames = hNoderef:ATTRIBUTE-NAMES.
REPEAT jx = 1 TO NUM-ENTRIES(anames):
  bname = ENTRY(jx, anames).
  MESSAGE "attribute-name is" bname "value is"
    hNoderef:GET-ATTRIBUTE(bname).
END.
```

GET-ATTRIBUTE-NODE( ) method

Returns the XML ATTRIBUTE node with the specified name.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
GET-ATTRIBUTE-NODE ( attr-node-handle , name )
```

attr-node-handle

A valid X-noderef handle to use for the XML ATTRIBUTE node.

name

A character expression representing the name of the XML ATTRIBUTE node. For a namespace-aware ATTRIBUTE node, you must qualify the node name with a prefix including a colon (for example, *prefix:node-name*).

GET-BINARY-DATA( ) method

Returns a MEMPTR containing the binary data in the file specified in the form field. The AVM sets the size of the MEMPTR to match the size of the file. This method is called by the `get-binary-data` WebSpeed API function. Intended for internal use only.

Return type: MEMPTR

Applies to: WEB-CONTEXT system handle

Syntax

```
GET-BINARY-DATA ( INPUT field-name )
```

field-name

The name of the form field containing the name of the file posted in the web request received by the WebSpeed Agent.

If the specified field is not part of the form, or the field is not of type 'file', the AVM returns the Unknown value (?) and displays an error message. You can suppress this message by using NO-ERROR on the statement containing the method.

GET-BLUE-VALUE( ) method
(Graphical interfaces only)

Returns the blue component of an entry in the color table.

Return type: INTEGER

Applies to: COLOR-TABLE system handle

Syntax

```
GET-BLUE-VALUE ( index )
```

index

An integer expression that specifies an entry in the color table.

GET-BROWSE-COLUMN( ) method

Returns the handle for the requested browse column.

Return type: HANDLE

Applies to: BROWSE widget

Syntax

```
GET-BROWSE-COLUMN ( col-index )
```

col-index

An integer value specifying the 1-based index into the browse column list.

GET-BUFFER-HANDLE( ) method

Gets the handle to a particular buffer of a query or ProDataSet object.

Return type: HANDLE

Applies to: ProDataSet object handle, Query object handle

Syntax

```
GET-BUFFER-HANDLE ( buffer-sequence-number | buffer-name )
```

buffer-sequence-number

An integer that represents the sequence number of the desired buffer.

Note: Sequence numbers for buffers of a query start at one, where one represents the top level and subsequent numbers represent lower levels of join, if any.

buffer-name

A CHARACTER expression that evaluates to the name of a buffer in the query or ProDataSet object.

If the buffer cannot be found, this attribute returns the Unknown value (?).

GET-BYTES-AVAILABLE( ) method

Indicates the number of bytes available for reading from the socket.

Return type: INTEGER

Applies to: Socket object handle

Syntax

```
GET-BYTES-AVAILABLE( )
```

GET-CALLBACK-PROC-CONTEXT( ) method

Returns the handle of the procedure that contains the internal procedure associated with the ABL callback for the specified event.

Return type: HANDLE

Applies to: Buffer object handle, ProDataSet object handle, Query object handle

Syntax

```
GET-CALLBACK-PROC-CONTEXT ( event-name )
```

event-name

The name of a defined event.

If the object does not have a callback procedure for the specified event, this method returns the Unknown value (?).

Use the SET-CALLBACK-PROCEDURE( ) method to associate an internal procedure with a callback for an object.

For more information on events, see the Handle-based Object Events Reference.

See also

APPLY-CALLBACK( ) method, GET-CALLBACK-PROC-NAME( ) method, SET-CALLBACK-PROCEDURE( ) method

GET-CALLBACK-PROC-NAME( ) method

Returns the name of the internal procedure associated with the ABL callback for the specified event.

Return type: CHARACTER

Applies to: Buffer object handle, ProDataSet object handle, Query object handle

Syntax

```
GET-CALLBACK-PROC-NAME ( event-name )
```

event-name

The name of a defined event.

If the object does not have a callback procedure for the specified event, this method returns the Unknown value (?).

Use the SET-CALLBACK-PROCEDURE( ) method to associate an internal procedure with a callback for an object.

For more information on events, see the Handle-based Object Events Reference.

See also

APPLY-CALLBACK( ) method, GET-CALLBACK-PROC-CONTEXT( ) method, SET-CALLBACK-PROCEDURE( ) method

GET-CGI-LIST( ) method

Gets the list of CGI environment variables. This method is called by the `get-cgi` WebSpeed API function. Intended for internal use only.

Return type: CHARACTER

Applies to: WEB-CONTEXT system handle

GET-CGI-VALUE( ) method

Gets the value of a specified CGI environment variable. This method is called by the `get-cgi` WebSpeed API function. Intended for internal use only.

Return type: CHARACTER

Applies to: WEB-CONTEXT system handle

GET-CGI-LONG-VALUE( ) method

Returns a LONGCHAR value in either the code page specified in the HTML-CHARSET attribute, if that code page is valid for a LONGCHAR, or `-cpinternal`. Otherwise, it returns the Unknown value (?) and displays an error message. You can suppress this message by using NO-ERROR on the statement containing the method. This method is called by the `get-cgi-long` and `get-long-value` WebSpeed API functions. Intended for internal use only.

Return type: LONGCHAR

Applies to: WEB-CONTEXT system handle

GET-CHANGES( ) method

Loads all the temp-tables or a single temp-table of an empty target ProDataSet object with changed rows from all temp-tables or a single temp-table (respectively) of the original (source) ProDataSet object.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle (of the target object)

Syntax

The syntax to invoke this method follows:

```
change-handle:GET-CHANGES ( original-handle [ , get-parent-mode ] )
```

change-handle

A handle to the target ProDataSet object or to the buffer object of the target ProDataSet temp-table to receive the changed rows.

original-handle

A handle to the original (source) ProDataSet object or a handle to the buffer object of a single source ProDataSet temp-table that contains the changed rows to load into the target object.

get-parent-mode

An optional logical expression, where TRUE indicates that the AVM gets the **changed** parent rows of each changed child row.

When TRUE, the AVM includes the parent row of each changed child row in the ProDataSet object or ProDataSet temp-table (if any). If there is more than one parent level above the changed row, the AVM includes the parent row at each level. In this case, the parent temp-tables must have a unique primary index that the AVM can use to find the corresponding rows. If a parent row has changed, the AVM copies both the before-image and after-image of the parent row. If a parent row has not changed, there will be no before-image of the parent row, and its change state (ROW-STATE) will be ROW-UNMODIFIED (0) or the Unknown value (?).

Note: When the relation mode of a parent is REPOSITION, no attempt is made to find that parent.

When FALSE, the AVM does not include parent rows. The default value is FALSE.

The ProDataSet objects associated with a target ProDataSet handle and the *original-handle* must have the same number of temp-table buffers, and the definition of the corresponding temp-tables must match (that is, in the number of columns, data types, and so on). Likewise, the temp-table objects associated with a target temp-table and the *original-handle* must match.

Once the changed rows are loaded, the AVM sets the ORIGIN-HANDLE attribute on the temp-tables in the receiving ProDataSet object to the corresponding temp-tables in the original source ProDataSet object. The AVM also sets the ORIGIN-ROWID attribute on each of the before-image table rows created in the receiving ProDataSet object to the ROWID of the corresponding before-image table row in the original source temp-table. The MERGE-CHANGES( ) method and MERGE-ROW-CHANGES( ) method uses these values to match up temp-tables and temp-table rows during a merge operation.

Note: When executing the method, there is no guarantee as to the contents of the buffers involved after the method finishes executing.

GET-CHILD( ) method

Retrieves a specific child node of the current node. The first parameter must be a valid X-noderef handle and will refer to the specified child XML node if the method succeeds.

Return type: LOGICAL

Applies to: X-document object handle, X-noderef object handle

Syntax

```
GET-CHILD ( x-node-handle , index )
```

x-node-handle

A valid X-noderef handle to use as the child XML node.

index

An integer representing the relative number in the node-tree (1 based).

The following code fragment demonstrates getting all the child nodes from the XML node referenced by hNoderef using the GET-CHILD( ) method:

```
. . .
REPEAT jx = 1 TO hNoderef:NUM-CHILDREN:
  ok = hNoderef:GET-CHILD(hNoderefChild, jx).
  IF NOT ok THEN LEAVE.
   . . .
END.
```

GET-CHILD-RELATION( ) method

Gets the handle to a data-relation object for which the buffer is the parent.

Return type: HANDLE

Applies to: Buffer object handle

Syntax

```
GET-CHILD-RELATION ( index )
```

index

An integer expression indicating the 1-based index of the data-relation object.

GET-CLIENT( ) method

Returns the handle to a copy of the sealed client-principal object that represents the user identity for the ABL session. If no identity has been established for the session using the SECURITY-POLICY:SET-CLIENT( ) method, this method returns the Unknown value (?).

Return type: HANDLE

Applies to: SECURITY-POLICY system handle

Syntax

```
GET-CLIENT ( )
```

You can use the client-principal object returned by this function to set the user's identity for other ABL sessions or database connections.

To avoid a memory leak, you must explicitly delete the client-principal object whose handle is returned by this method when you no longer need it.

This method does not raise an ERROR condition unless it encounters an unexpected internal error.

See also

Client-principal object handle, GET-DB-CLIENT function, SET-CLIENT( ) method, SETUSERID function

GET-COLUMN( ) method

Returns the column number at which the *n*th message occurred during the preceding compilation.

Return type: INTEGER

Applies to: COMPILER system handle

Syntax

```
GET-COLUMN ( n )
```

n

An integer expression, from 1 to the value of the NUM-MESSAGES attribute, that specifies the numeric occurrence of a compilation message. If the *n*th occurrence of a compilation message did not occur, the compiler issues a warning message at runtime.

GET-CONFIG-VALUE( ) method

Gets the value of parameters set in the WebSpeed configuration file. This method is called by the `get-config` WebSpeed API function. Intended for internal use only.

Return type: CHARACTER

Applies to: WEB-CONTEXT system handle

GET-CURRENT( ) method

Refetches the current record or records associated with the query.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
GET-CURRENT ( NO-LOCK | SHARE-LOCK [ , NO-WAIT ]
 | EXCLUSIVE-LOCK [ , NO-WAIT ]  )
```

NO-LOCK

Specifies that no lock is applied to the record. This applies to all buffers in a join.

SHARE-LOCK

Specifies that the record is share locked. This applies to all buffers in a join.

EXCLUSIVE-LOCK

Specifies that the record is exclusively locked. This applies to all buffers in a join.

NO-WAIT

Specifies that the method returns immediately if the record cannot be accessed because it is locked by another user. If you do not use the NO-WAIT option, the method waits until the record can be accessed. This applies to all buffers in a join. If you specify NO-WAIT and the record is locked by another user, the record is returned to you with NO-LOCK and the LOCKED function returns TRUE for the record.

GET-DATASET-BUFFER( ) method

Gets the handle to the ProDataSet object buffer associated with the data-source object.

Return type: HANDLE

Applies to: Data-source object handle

Syntax

```
GET-DATASET-BUFFER ( )
```

GET-DOCUMENT-ELEMENT( ) method

Retrieves the root element of the document. The parameter must be a valid X-noderef handle and will refer to the document's root element if the method succeeds.

Return type: LOGICAL

Applies to: X-document object handle

Syntax

```
GET-DOCUMENT-ELEMENT ( x-node-handle )
```

x-node-handle

A valid X-noderef handle to use for the root element.

The following example demonstrates the use of GET-DOCUMENT-ELEMENT if hDoc is an X-document and hRoot is an X-noderef:

```
/* Creates an ABL document object & initializes the associated XML
   object. */
CREATE X-document hDoc.

/* Creates an ABL reference for an XML node in a parse tree. */
CREATE X-NODEREF hRoot.

/* Reads the myxml.xml document into an XML parse tree. */
hDoc:LOAD("file","myxml.xml",false).

/* Associates hRoot with the root node of the hDoc document. */
hDoc:GET-DOCUMENT-ELEMENT(hRoot).
```

GET-DROPPED-FILE( ) method
(Windows only; Graphical interfaces only)

Returns the name of the dropped file indicated by the *index* parameter.

Return type: CHARACTER

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

Syntax

```
GET-DROPPED-FILE ( index )
```

index

An integer expression that specifies the ordinal number of a dropped file. If *index* is less than 1 or greater than the number of dropped files (as indicated by NUM-DROPPED-FILES), GET-DROPPED-FILE( ) returns the Unknown value (?). If there is no current drag-and-drop operation, GET-DROPPED-FILE( ) returns the Unknown value (?).

The returned filename can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

GET-DYNAMIC( ) method
(Graphical interfaces only)

Returns TRUE if the entry in the color table is a dynamic color.

Return type: LOGICAL

Applies to: COLOR-TABLE system handle

Syntax

```
GET-DYNAMIC ( index )
```

index

An integer expression that specifies an entry in the color table.

GET-ERROR-COLUMN( ) method

Returns the listing column number at which the *n*th error occurred in the preceding ABL source code compilation.

This method is supported only for backward compatibility. Use the GET-COLUMN( ) method instead.

Return type: INTEGER

Applies to: COMPILER system handle

Syntax

```
GET-ERROR-COLUMN ( n )
```

n

An integer expression, from 1 to the value of the NUM-MESSAGES attribute, that specifies the numeric occurrence of a compilation error. If the *n*th occurrence of a compilation error did not occur, this method returns the Unknown value (?).

GET-ERROR-ROW( ) method

Returns the listing line number at which the *n*th error occurred in the preceding ABL source code compilation.

This method is supported only for backward compatibility. Use the GET-ROW( ) method instead.

Return type: INTEGER

Applies to: COMPILER system handle

Syntax

```
GET-ERROR-ROW ( n )
```

n

An integer expression, from 1 to the value of the NUM-MESSAGES attribute, that specifies the numeric occurrence of a compilation error.

If the *n*th occurrence of a compilation error did not occur, this method returns the Unknown value (?).

GET-FILE-NAME( ) method

Returns the name of the source file associated with the *n*th error that occurred in the preceding ABL source code compilation.

Return type: CHARACTER

Applies to: COMPILER system handle

Syntax

```
GET-FILE-NAME ( n )
```

n

An integer expression, from 1 to the value of the NUM-MESSAGES attribute, that specifies the numeric occurrence of a compilation error.

If the *n*th occurrence of a compilation error did not occur, this method returns the Unknown value (?).

GET-FILE-OFFSET( ) method

Returns the character offset at which the *n*th error occurred in the associated source file during the preceding ABL source code compilation.

Return type: INTEGER

Applies to: COMPILER system handle

Syntax

```
GET-FILE-OFFSET ( n )
```

n

An integer expression, from 1 to the value of the NUM-MESSAGES attribute, that specifies the numeric occurrence of a compilation error.

If the *n*th occurrence of a compilation error did not occur, this method returns the Unknown value (?).

GET-FIRST( ) method

Moves a query object's result list pointer to the first row.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
GET-FIRST ( NO-LOCK | SHARE-LOCK [ , NO-WAIT ]
| EXCLUSIVE-LOCK [ , NO-WAIT ]  )
```

NO-LOCK

Specifies that no lock is applied to the record. This applies to all buffers in a join. Unless explicitly specified otherwise, this is the default lock type for this method.

SHARE-LOCK

Specifies that the record is share locked. This applies to all buffers in a join.

EXCLUSIVE-LOCK

Specifies that the record is exclusively locked. This applies to all buffers in a join.

NO-WAIT

Specifies that the method returns immediately if the record cannot be accessed because it is locked by another user. If you do not use the NO-WAIT option, the method waits until the record can be accessed. This applies to all buffers in a join. If you specify NO-WAIT and the record is locked by another user, the record is returned to you with NO-LOCK and the LOCKED function returns TRUE for the record.

GET-FIRST() returns TRUE if the first record in the query is found. If the query is not open or the first record cannot be found (query is empty), the method returns FALSE. If the query object handle is invalid, an error is raised.

A query that includes a BREAK BY phrase becomes a FORWARD-ONLY query. In this case you cannot use the GET-FIRST() method. If you do, the AVM raises ERROR.

GET-GREEN-VALUE( ) method
(Graphical interfaces only)

Returns the green component of an entry in the color table.

Return type: INTEGER

Applies to: COLOR-TABLE system handle

Syntax

```
GET-GREEN-VALUE ( index )
```

index

An integer expression that specifies an entry in the color table.

GET-HEADER-ENTRY( ) method

Retrieves the SOAP-header-entryref object at the given header entry index (base 1). In other words, a SOAP-header object contains a list of entries. Each entry points to an existing SOAP-header-entryref object. This method finds the SOAP-header-entryref object associated with indexed header entry and copies its handle to the *header-entryref* handle you provided.

Return type: LOGICAL

Applies to: SOAP-header object handle

Syntax

```
GET-HEADER-ENTRY ( header-entryref , index )
```

header-entryref

A valid SOAP-header-entryref object handle.

index

An integer expression indicating the 1-based index of the header entry.

GET-INDEX-BY-NAMESPACE-NAME( ) method

Gets the 1-based index of the attribute with the given namespace name.

Return type: INTEGER

Applies to: SAX-attributes object handle

Syntax

```
GET-INDEX-BY-NAMESPACE-NAME ( uri , localname )
```

uri

The namespace URI (that is, the URI to which the attribute's prefix refers), or if the name has no namespace URI, an empty string.

localname

The local (unqualified) name of the attribute.

Returns the Unknown value (?) if no attribute's uri-localname combination matches the combination of *uri* and *localname*.

GET-INDEX-BY-QNAME( ) method

Gets the 1-based index of the attribute with the given XML qualified name.

Return type: INTEGER

Applies to: SAX-attributes object handle

Syntax

```
GET-INDEX-BY-QNAME ( qname )
```

qname

A CHARACTER expression indicating the XML qualified name of the attribute of interest.

Returns the Unknown value (?) if no attribute's XML qualified name matches *qname.*

GET-ITERATION( ) method (Data Objects)

Returns the buffer handle at a given level of iteration during a recursive FILL of a ProDataSet.

Return type: HANDLE

Applies to: Buffer object handle

Syntax

```
GET-ITERATION ( level )
```

level

An integer expression indicating the level of iteration.

As a recursive ProDataSet FILL is proceeding, it creates clones of the relevant buffers, relations, queries and data-sources for each level of recursion. As a new record is added to the ProDataSet, it fires FILL events on the recursed, cloned buffer. From inside the event handler, you may want to see previous iterations of the buffer; its parent, grandparent, great-grandparent, and so on. GET-ITERATION( ) returns the buffer handle at a specified recursion level.

If the buffer object handle does not point to a ProDataSet temp-table or is not part of a recursive data-relation, GET-ITERATION returns the Unknown value (?).

See also

CURRENT-ITERATION attribute (Data Objects), FILL( ) method, NUM-ITERATIONS attribute (data objects), RECURSIVE attribute

---

GET-ITERATION( ) method (Widget Objects)

Returns the handle for the field group that represents the *n*th visible iteration of the frame.

Return type: HANDLE

Applies to: FRAME widget

Syntax

```
GET-ITERATION ( n )
```

n

An integer expression that specifies the number of a visible frame iteration.

You can read the NUM-ITERATIONS attribute of the frame to determine how many visible foreground (data) iterations the frame contains. You can then use the FIRST-CHILD or LAST-CHILD attributes of the field group to access the first or last field-level widget (respectively) in the iteration.

GET-LAST( ) method

Moves a query object's result list pointer to the last row.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
GET-LAST ( NO-LOCK | SHARE-LOCK [ , NO-WAIT ]
 | EXCLUSIVE-LOCK [ , NO-WAIT ]  )
```

NO-LOCK

Specifies that no lock is applied to the record. This applies to all buffers in a join. Unless explicitly specified otherwise, this is the default lock type for this method.

SHARE-LOCK

Specifies that the record is share locked. This applies to all buffers in a join.

EXCLUSIVE-LOCK

Specifies that the record is exclusively locked. This applies to all buffers in a join.

NO-WAIT

Specifies that the method returns immediately if the record cannot be accessed because it is locked by another user. If you do not use the NO-WAIT option, the method waits until the record can be accessed. This applies to all buffers in a join. If you specify NO-WAIT and the record is locked by another user, the record is returned to you with NO-LOCK and the LOCKED function returns TRUE for the record.

GET-LAST() returns TRUE if the last record in the query is found. If the query is not open or the last record cannot be found (query is empty), the method returns FALSE. If the query object handle is invalid, an error is raised.

A query that includes a BREAK BY phrase becomes a FORWARD-ONLY query. In this case you cannot use the GET-LAST() method. If you do, the AVM raises ERROR.

GET-LOCALNAME-BY-INDEX( ) method

Gets the local (unqualified) name of the attribute at the given 1-based index.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-LOCALNAME-BY-INDEX ( index )
```

index

An integer expression indicating the 1-based index of the attribute.

Looks up an attribute's local (unqualified) name by index. Returns the Unknown value (?) if namespace processing is disabled, if the index is less than 1, or if the index is greater than the value of the SAX-attributes object's NUM-ITEMS attribute.

GET-MESSAGE( ) method (Handle)

Returns the message associated with a specified error currently recorded by the specified system handle.

Return type: CHARACTER

Applies to: COMPILER system handle, ERROR-STATUS system handle

Syntax

```
GET-MESSAGE ( n )
```

n

An integer expression, from 1 to the value of the NUM-MESSAGES attribute, that specifies the numeric occurrence of the error whose information you want to retrieve.

For the COMPILER system handle, this is the message returned for the *n*th error that occurred during the preceding ABL source code compilation.

For the ERROR-STATUS system handle, this is the message returned for the *n*th error that occurred during the execution of a statement run with the NO-ERROR option.

GET-MESSAGE-TYPE( ) method

Returns the message type associated with the *n*th message that occurred during the preceding compilation.

Return type: INTEGER

Applies to: COMPILER system handle

Syntax

```
GET-MESSAGE-TYPE ( n )
```

n

An integer expression, from 1 to the value of the NUM-MESSAGES attribute, that specifies the numeric occurrence of a compilation message. If the *n*th occurrence of a compilation message did not occur, the compiler issues a warning message at runtime.

The GET-MESSAGE-TYPE method returns one of the following values:

- **1** — Error message
- **2** — Warning message

- **3** — Preprocessor directive

GET-NEXT( ) method

Moves a query object's result list pointer ahead one row.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
GET-NEXT ( NO-LOCK | SHARE-LOCK [ , NO-WAIT ]
 | EXCLUSIVE-LOCK [ , NO-WAIT ]  )
```

`NO-LOCK`

Specifies that no lock is applied to the record. This applies to all buffers in a join. Unless explicitly specified otherwise, this is the default lock type for this method.

`SHARE-LOCK`

Specifies that the record is share locked. This applies to all buffers in a join.

`EXCLUSIVE-LOCK`

Specifies that the record is exclusively locked. This applies to all buffers in a join.

`NO-WAIT`

Specifies that the method returns immediately if the record cannot be accessed because it is locked by another user. If you do not use the NO-WAIT option, the method waits until the record can be accessed. This applies to all buffers in a join. If you specify NO-WAIT and the record is locked by another user, the record is returned to you with NO-LOCK and the LOCKED function returns TRUE for the record.

GET-NEXT() returns TRUE if the next record in the query is found. If the query is not open or the next record cannot be found (query is empty or the query result list pointer is on the last row), the method returns FALSE. If the query object handle is invalid, an error is raised.

GET-NODE( ) method

Returns a handle to an X-noderef object that refers to the XML underlying a SOAP-header-entryref object or SOAP fault entry.

Note: A SOAP-header object contains a list of entries that each point to an exiting SOAP-header-entryref object. It is a SOAP-header-entryref object that the method is operating on.

Return type: LOGICAL

Applies to: SOAP-fault-detail object handle, SOAP-header-entryref object handle

Syntax

```
GET-NODE ( x-noderef )
```

x-noderef

A variable of type X-noderef that refers to the root node of a DOM tree that has a SOAP header entry or SOAP fault entry as its root.

The X-noderef will have namespace declarations for all namespaces that are in effect for the SOAP header entry element. The X-noderef will include attributes for all attributes that the SOAP header entry has, including SOAP ENV:mustUnderstand and SOAP-ENV:Actor. Operations performed on the X NODEREF, its child X-noderefs, will directly affect the underlying header entry (note that this contrasts with the LONGCHAR returned from GET-SERIALIZED( ) method).

Returns the Unknown value (?) if the SOAP-header-entryref object handle has been initialized but does not refer to a header entry (for example, immediately after the CREATE SOAP-HEADER-ENTRYREF statement).

GET-NUMBER( ) method

Returns the error number associated with a specified error currently recorded by the specified system handle.

Return type: INTEGER

Applies to: COMPILER system handle, ERROR-STATUS system handle

Syntax

```
GET-NUMBER ( n )
```

n

An integer expression, from 1 to the value of the NUM-MESSAGES attribute, that specifies the numeric occurrence of the error whose information you want to retrieve.

For the COMPILER system handle, this is the error number returned for the *n*th error that occurred during the preceding ABL source code compilation.

For the ERROR-STATUS system handle, this is the error number returned for the nth error that occurred during the execution of a statement run with the NO-ERROR option.

GET-PARENT( ) method

Retrieve the parent node of the node. The first parameter must be a valid X-noderef handle and will refer to the parent XML node if the node has a parent. If the node is the top "root" element in the document, the method will fail, and the X-noderef handle will not refer to a node.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
GET-PARENT ( x-node-handle )
```

x-node-handle

A valid X-noderef handle to use for the parent XML node.

The following example returns a handle to the parent XML node in hNoderefParent unless the hNoderef is the top "root" element in the hDoc.

```
hNoderef:GET-PARENT(hNoderefParent)
```

GET-PREV( ) method

Moves a query object's result list pointer back one row.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
GET-NEXT ( NO-LOCK | SHARE-LOCK [ , NO-WAIT ]
| EXCLUSIVE-LOCK [ , NO-WAIT ]  )
```

NO-LOCK

Specifies that no lock is applied to the record. This applies to all buffers in a join. Unless explicitly specified otherwise, this is the default lock type for this method.

SHARE-LOCK

Specifies that the record is share locked. This applies to all buffers in a join.

EXCLUSIVE-LOCK

Specifies that the record is exclusively locked. This applies to all buffers in a join.

NO-WAIT

Specifies that the method returns immediately if the record cannot be accessed because it is locked by another user. If you do not use the NO-WAIT option, the method waits until the record can be accessed. This applies to all buffers in a join. If you specify NO-WAIT and the record is locked by

another user, the record is returned to you with NO-LOCK and the LOCKED function returns TRUE for the record.

GET-PREV() returns TRUE if the previous record in the query is found. If the query is not open or the previous record cannot be found (query is empty or the query result list pointer is on the first row), the method returns FALSE. If the query object handle is invalid, an error is raised.

A query that includes a BREAK BY phrase becomes a FORWARD-ONLY query. In this case you cannot use the GET-PREV() method. If you do, the AVM raises ERROR.

GET-PROPERTY( ) method

Gets the value of the specified application-defined property stored in the client-principal object. The client-principal object may be sealed or unsealed. If the specified property is not stored in the object, or the property does not have a value, this method returns the Unknown value (?).

Return type: CHARACTER

Applies to: Client-principal object handle

Syntax

```
GET-PROPERTY ( property-name )
```

property-name

A case-sensitive character string that specifies the name of an application-defined property stored in the client-principal object. You must enclose this character string in quotes.

You can also use the LIST-PROPERTY-NAMES( ) method to retrieve a list of all application-defined properties stored in the client-principal.

The following code fragment illustrates how to use the GET-PROPERTY( ) method:

```
DEFINE VARIABLE hCP  AS HANDLE    NO-UNDO.
DEFINE VARIABLE vVal AS CHARACTER NO-UNDO.
. . .
CREATE CLIENT-PRINCIPAL hCp.
. . .
vVal = hCP:GET-PROPERTY("eye-color").
DISPLAY "Eye color: " vVal.
```

See also

LIST-PROPERTY-NAMES( ) method, SET-PROPERTY( ) method

GET-PRINTERS( ) method
(Windows only)

Returns a comma-separated list of printers defined in the Windows Registry.

Return type: CHARACTER

Applies to: SESSION system handle

Syntax

```
SESSION:GET-PRINTERS( )
```

If there are no printers defined in the Windows Registry, this method returns the null string (""). Network printers appear in Universal Naming Convention format.

GET-QNAME-BY-INDEX( ) method

Gets the XML qualified name of the attribute at the given 1-based index.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-QNAME-BY-INDEX ( index )
```

index

An integer expression indicating the 1-based index of the attribute.

Looks up an attribute's XML qualified name by index. Returns Unknown value (?) if the index is less than 1 or greater than the value of the SAX-attributes object's NUM-ITEMS attribute.

GET-RED-VALUE( ) method
(Graphical interfaces only)

Returns the red component of an entry in the color table.

Return type: INTEGER

Applies to: COLOR-TABLE system handle

Syntax

```
GET-RED-VALUE ( index )
```

index

An integer expression that specifies an entry in the color table.

GET-RELATION( ) method

Gets the handle of the specified data-relation object.

Return type: HANDLE

Applies to: ProDataSet object handle

Syntax

```
GET-RELATION ( index | relation-name )
```

index

An integer expression indicating the 1-based index of the data-relation object.

relation-name

A character expression that evaluates to the name of the data-relation object.

GET-REPOSITIONED-ROW( ) method

Returns the row index of the browse viewport where the REPOSITION TO ROWID (or RECID) statement displays a repositioned record.

Return type: INTEGER

Applies to: BROWSE widget

Syntax

```
GET-REPOSITIONED-ROW ( )
```

By default, this is the top row in the browse viewport (index 1). Note that this method is only useful in conjunction with the REPOSITION statement.

See the SET-REPOSITIONED-ROW( ) method reference entry for more information.

GET-RGB-VALUE( ) method
(Graphical interfaces only)

Returns an INTEGER that represents a combination of the red, green, and blue value of an entry in the color table.

Return type: INTEGER

Applies to: COLOR-TABLE system handle

Syntax

```
GET-RGB-VALUE ( index )
```

index

An integer expression that specifies an entry in the color table.

Note: This method is useful primarily when using colors with ActiveX controls.

GET-ROW( ) method

Returns the line number at which the *n*th message occurred during the preceding compilation.

Return type: INTEGER

Applies to: COMPILER system handle

Syntax

```
GET-ROW ( n )
```

n

An integer expression, from 1 to the value of the NUM-MESSAGES attribute, that specifies the numeric occurrence of a compilation message. If the *n*th occurrence of a compilation message did not occur, the compiler issues a warning message at runtime.

GET-SAFE-USER( ) method

This method only applies to a Progress Application Server (PAS) for OpenEdge application server.

Returns the handle to a copy of the sealed client-principal object that represents the safe user identity for the specified multi-tenant database connection. The application is responsible for deleting the handle object when done. If the Safe User ID feature is disabled for the specified database, the method returns the Unknown (?) value.

Return-type: HANDLE

Applies to: SECURITY-POLICY system handle

Syntax

```
GET-SAFE-USER ( logical-db-name )
```

logical-db-name

The logical name of the database from which you want to retrieve the safe user identity. The logical database name must be a character string enclosed in quotes, or a character expression.

The following example code shows how to call the GET-SAFE-USER( ) method.

```
DEFINE VARIABLE hCP AS HANDLE NO-UNDO.

hCP = SECURITY-POLICY:GET-SAFE-USER(LDBNAME(1)).
IF hCP = ?
THEN
  MESSAGE "Database is not configured to use the Safe User ID feature".
ELSE DO:
  MESSAGE "Database is configured to use the Safe User ID feature".
  DELETE OBJECT hCP.  // clean up the object
END.
```

See also

SET-SAFE-USER( ) method

Overview of Safe User ID in *Learn about Identity Management*

GET-SELECTED-WIDGET( ) method

Returns the handle of the selected widget in a dialog box, frame, or window.

Return type: HANDLE

Applies to: DIALOG-BOX widget, FRAME widget, WINDOW widget

Syntax

```
GET-SELECTED-WIDGET ( n )
```

n

An integer expression that specifies an index to a selected widget in a frame, dialog box, or window.

You can use the NUM-SELECTED-WIDGETS attribute to determine the total number of selected widgets within the frame or window. The order of the selected widgets is unpredictable.

GET-SERIALIZED( ) method

Returns a LONGCHAR that contains the serialized form of the XML underlying the SOAP-header-entryref or SOAP fault entry. (The SOAP-header object contains a list of entries that each point to an exiting SOAP-header-entryref object. It is the SOAP-header-entryref object that the method serializes.)

Return type: LONGCHAR

Applies to: SOAP-fault-detail object handle, SOAP-header-entryref object handle

Syntax

```
GET-SERIALIZED ( )
```

Contents of the returned LONGCHAR are equivalent to taking the X-noderef returned by the GET-NODE( ) method and serializing the data. The caller of this method is responsible for managing the lifetime of the LONGCHAR. Changes made to the contents of the LONGCHAR will NOT be reflected in the original header entry unless it is passed as the parameter in the SET-SERIALIZED( ) method.

Returns the Unknown value (?) if the SOAP-header-entryref object handle has been initialized but does not refer to a header entry (for example, immediately after the CREATE SOAP-HEADER-ENTRYREF statement).

GET-SIGNATURE( ) method

Returns the *signature* of the internal procedure or user-defined function whose name you supply. Specifically:

- If you provide the name of an internal procedure, GET-SIGNATURE returns the type and mode of each parameter.
- If you provide the name of a user-defined function, GET-SIGNATURE returns the return type, and the type and mode of each parameter.
- If you provide the nil procedure name (""), GET-SIGNATURE returns the signature of the procedure whose handle you supply.
- If you provide a name that does not match any of the internal procedures or user-defined functions in the procedure, GET-SIGNATURE returns the empty string ("").
- If you provide a remote (proxy) procedure handle or the name of a Web service procedure, GET-SIGNATURE returns the empty string ("").
- If you provide the name of a DLL entry point, GET-SIGNATURE returns the ABL equivalent of the C data type of each parameter of the entry point. For more information, see *OpenEdge Programming Interfaces*.

Note: GET-SIGNATURE does not return the signature of any internal procedure defined using the PROCEDURE statement's PRIVATE option. Similarly, GET-SIGNATURE does not return the signature of any user-defined function defined using the FUNCTION statement's PRIVATE option.

Return type: CHARACTER

Applies to: Procedure object handle

Syntax

```
GET-SIGNATURE ( int-proc-name )
```

int-proc-name

The name of an internal procedure or user-defined function.

GET-SIGNATURE returns a string with the following format:

```
type , return-type , [modenamep-type[ , modenamep-type]...]
```

type

The type of the internal procedure. Types include:

- **PROCEDURE** — An ABL internal procedure
- **FUNCTION** — An ABL user-defined function whose definition resides in the procedure
- **EXTERN** — An ABL user-defined function whose definition resides in another procedure
- **DLL-ENTRY** — A DLL entry point
- **MAIN** — The main procedure

return-type

(User-defined functions only) The ABL data type that a user-defined function returns.

mode name p-type

A parameter description where *mode* is the mode of the parameter, *name* is the name of the parameter, and *p-type* is the type of parameter. The parameter type is either a data type (scalar or array) or, for a buffer parameter, the name of the table associated with the buffer.

The modes are:

- INPUT
- OUTPUT
- INPUT-OUTPUT
- BUFFER
- INPUT TABLE
- OUTPUT TABLE
- INPUT-OUTPUT TABLE

The data types are:

- CHARACTER
- DATE
- DATETIME
- DATETIME-TZ
- DECIMAL
- HANDLE
- INT64
- INTEGER
- LOGICAL
- MEMPTR

- RAW
- RECID
- ROWID

When you define the parameter type as a determinate array with a constant extent value, the GET-SIGNATURE( ) method returns the constant extent value specified as part of the signature (for example, EXTENT 100). When you define the parameter type as a determinate array with a variable extent value, or as an indeterminate array, the GET-SIGNATURE( ) method returns only the extent keyword (that is, EXTENT, with no extent value).

GET-SOCKET-OPTION( ) method

Returns a comma separated string containing values appropriate for the specified socket option. Otherwise, it returns the Unknown value (?).

Return type: CHARACTER

Applies to: Socket object handle

Syntax

```
GET-SOCKET-OPTION ( name )
```

name

A character expression indicating the name of the socket option to be retrieved. The following table describes the options ABL supports.

Table 81: Options for GET-SOCKET-OPTION( )

Option	Value returned
TCP-NODELAY	An *enable indicator*, which is either TRUE or FALSE.
SO-LINGER	Two comma separated values: • The *onoff indicator*, which is either TRUE or FALSE. • The *linger time*. If the *onoff indicator* is FALSE, the linger time does not need to be provided.
SO-KEEPALIVE	TRUE if the option is on; FALSE otherwise.The default depends on how the socket object was created: • For socket objects created using CREATE SOCKET, the default is off. • For socket objects created by ABL and passed as a parameter to the event-procedure context, the default is on.
SO-REUSEADDR	TRUE if the option is on; FALSE otherwise.The default depends on the platform.

Option	Value returned
SO-RCVBUF	An integer that indicates the size of the receive buffer.The default depends on the platform.
SO-SNDBUF	An integer that indicates the size of the send buffer.The default depends on the platform.
SO-RCVTIMEO	The timeout length—that is, the number of seconds you want the socket to wait for expected data before timing out.The default is -1, which tells the socket to wait forever. **Note:** The timeout length is not guaranteed to be precise to the second.

This method returns option-specific data if the retrieval of the option succeeded and the Unknown value (?) otherwise. An error can occur if:

- The value of *name* is not an ABL-supported socket option
- Getting the socket option fails

GET-SOURCE-BUFFER( ) method

Gets the handle to the source buffer in the data-source object at the specified index position.

Return type: HANDLE

Applies to: Data-source object handle

Syntax

```
GET-SOURCE-BUFFER ( [ buffer-index ] )
```

buffer-index

An optional integer expression indicating the 1-based index of the source buffer. The default is 1.

GET-TAB-ITEM( ) method

Returns the handle of a widget at a specified tab position in a field group.

Return type: HANDLE

Applies to: FIELD-GROUP widget

Syntax

```
GET-TAB-ITEM ( n )
```

n

An integer expression that specifies a tab position within a field group.

You can use the MOVE-AFTER-TAB-ITEM( ) and MOVE-BEFORE-TAB-ITEM( ) methods to change the tab position of fields at the field level, and the FIRST-TAB-ITEM and LAST-TAB-ITEM attributes to change the tab positions at the field group level.

If the widget returned is a frame, the specified tab position includes the tab positions of all tab-order widgets contained by the frame. For more information on how frames owned by a field group participate in the tab order of that field group, see the FRAME widget reference entry.

GET-TEXT-HEIGHT-CHARS( ) method
(Graphical interfaces only)

Returns the height, in character units, of the specified font. If no font is specified, the method returns the height of the default font.

Return type: DECIMAL

Applies to: FONT-TABLE system handle

Syntax

```
GET-TEXT-HEIGHT-CHARS ( [font] )
```

font

An integer expression that specifies an entry within the font.

If you pass the Unknown value (?) to this method, the AVM uses the system default font. When a field-level widget inherits its font from the parent frame, the AVM returns the Unknown value (?) for the font and you must use the font of the parent frame.

GET-TEXT-HEIGHT-PIXELS( ) method
(Graphical interfaces only)

Returns the height, in pixels, of the specified font. If no font is specified, the method returns the height of the default font.

Return type: INTEGER

Applies to: FONT-TABLE system handle

Syntax

```
GET-TEXT-HEIGHT-PIXELS ( [ font ] )
```

font

An integer expression that specifies an entry within the font table.

If you pass the Unknown value (?) to this method, the AVM uses the system default font. When a field-level widget inherits its font from the parent frame, the AVM returns the Unknown value (?) for the font and you must use the font of the parent frame.

GET-TEXT-WIDTH-CHARS( ) method
(Graphical interfaces only)

Returns the width, in character units, of the string using the specified font. If no font is specified, the method calculates the width of the string using the default font.

Return type: DECIMAL

Applies to: FONT-TABLE system handle

Syntax

```
GET-TEXT-WIDTH-CHARS ( string [ , font ] )
```

string

A character-string expression whose width you want to determine.

font

An integer expression that specifies an entry within the font table.

If you pass the Unknown value (?) to this method, the AVM uses the system default font. When a field-level widget inherits its font from the parent frame, the AVM returns the Unknown value (?) for the font and you must use the font of the parent frame.

GET-TEXT-WIDTH-PIXELS( ) method
(Graphical interfaces only)

Returns the width, in pixels, of the string using the specified font. If no font is specified, the method calculates the width of the string using the default font.

Return type: INTEGER

Applies to: FONT-TABLE system handle

Syntax

```
GET-TEXT-WIDTH-PIXELS ( string [ , font ] )
```

string

A character-string expression whose width you want to determine.

font

An integer expression that specifies an entry within the font table.

If you pass the Unknown value (?) to this method, the AVM uses the system default font. When a field-level widget inherits its font from the parent frame, the AVM returns the Unknown value (?) for the font and you must use the font of the parent frame.

GET-TOP-BUFFER( ) method

Gets the top-level buffer in a ProDataSet object at the specified index position.

Note: A top-level buffer is a ProDataSet object buffer that is not a child in any active data relation. There may be one or more top-level buffers in a ProDataSet object.

Return type: HANDLE

Applies to: ProDataSet object handle

Syntax

```
GET-TOP-BUFFER ( index )
```

index

An integer expression indicating the 1-based index of the top-level buffer.

GET-TYPE-BY-INDEX( ) method

Gets the type of the attribute at the given 1-based index.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-TYPE-BY-INDEX ( index )
```

index

An integer expression indicating the 1-based index of the attribute.

The attribute type is one of the following strings: "CDATA," "ID," "IDREF," "IDREFS," "NMTOKEN," "NMTOKENS," "ENTITY," "ENTITIES," or "NOTATION." These are always uppercase.

Returns the Unknown value (?) if the index is less than 1 or greater than the value of SAX-attributes object's NUM-ITEMS attribute.

GET-TYPE-BY-NAMESPACE-NAME( ) method

Gets the type of the attribute with the given namespace name.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-TYPE-BY-NAMESPACE-NAME ( uri , localname )
```

uri

The namespace URI (that is, the URI to which the attribute's prefix refers), or, if the name has no namespace URI, an empty string.

localname

The local (unqualified) name of the attribute.

The attribute type is one of the following strings: "CDATA," "ID," "IDREF," "IDREFS," "NMTOKEN," "NMTOKENS," "ENTITY," "ENTITIES," or "NOTATION." These are always uppercase.

Returns the Unknown value (?) if no attribute's XML qualified name matches the combination of *uri* and *localname* or if namespace processing is disabled.

GET-TYPE-BY-QNAME( ) method

Gets the type of the attribute with the given XML qualified name.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-TYPE-BY-QNAME ( qname )
```

qname

A CHARACTER expression indicating the XML qualified name of the attribute of interest.

The attribute type is one of the following strings: "CDATA," "ID," "IDREF," "IDREFS," "NMTOKEN," "NMTOKENS," "ENTITY," "ENTITIES," or "NOTATION." These are always uppercase.

Returns the Unknown value (?) if no attribute's XML qualified name matches *qname*.

GET-URI-BY-INDEX( ) method

Gets the namespace URI of the attribute at the given 1-based index.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-URI-BY-INDEX ( index )
```

index

An integer expression indicating the 1-based index of the attribute.

Returns the Unknown value (?) if *index* is less than 1 or greater than the value of the SAX-attributes object's NUM-ITEMS attribute.

GET-VALUE-BY-INDEX( ) method

Gets the value of the attribute at the given 1-based index.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-VALUE-BY-INDEX ( index )
```

index

An integer expression indicating the 1-based index of the attribute.

Returns the Unknown value (?) if *index* is less than 1 or greater than the value of the SAX-attributes object's NUM-ITEMS attribute.

GET-VALUE-BY-NAMESPACE-NAME( ) method

Gets the value of the attribute with the given namespace name.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-VALUE-BY-NAMESPACE-NAME ( uri , localname )
```

uri

The namespace URI (that is, the URI to which the attribute's prefix refers), or, if the name has no namespace URI, an empty string.

localname

The local (unqualified) name of the attribute.

Returns the Unknown value (?) if no attribute has a namespace name that matches the *uri* and *localname*, or if namespace processing is disabled.

GET-VALUE-BY-QNAME( ) method

Gets the value of the attribute with the given XML qualified name.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-VALUE-BY-QNAME ( qname )
```

qname

A CHARACTER expression indicating the XML qualified name of the attribute.

Returns the Unknown value (?) if no attribute has a XML qualified name that matches *qname*.

GET-WAIT-STATE( ) method

Returns a string indicating the current wait-state.

Return type: CHARACTER

Applies to: SESSION system handle

Syntax

```
GET-WAIT-STATE ( )
```

If the SET-WAIT-STATE( ) method was called with "GENERAL" or "COMPILER", GET-WAIT-STATE( ) returns that string. If the SET-WAIT-STATE( ) method was called with an arbitrary pointer name, GET-WAIT-STATE( ) returns "CUSTOM". If there is no current wait-state, GET-WAIT-STATE( ) returns an empty string ("").

GRAPHIC-EDGE attribute
(Character interfaces only)

Indicates whether to draw a rectangle with graphic characters.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: RECTANGLE widget

When the GRAPHIC-EDGE attribute is TRUE, the rectangle is drawn with graphic characters, and the EDGE-CHARS and EDGE-PIXELS attributes are ignored.

GRID-FACTOR-HORIZONTAL attribute
(Graphical interfaces only)

The spacing, in horizontal grid units, between the horizontal grid lines of the frame.

Data type: INTEGER

Access: Readable/Writeable

Applies to: FRAME widget

If the value is 1, each horizontal grid unit is intersected by a line of vertical grid points. If the value is greater than 1, every *n*th horizontal grid unit is intersected by a line of vertical grid points, where *n* is the value of GRID-FACTOR-HORIZONTAL. The default value is 6.

The width of a horizontal grid unit is defined by the GRID-UNIT-WIDTH-CHARS or GRID-UNIT-WIDTH-PIXELS attribute.

Note: Setting this attribute to 1 has the same effect as setting the GRID-FACTOR-VERTICAL attribute to 1, because either setting makes all grid points visible in the frame.

GRID-FACTOR-VERTICAL attribute
(Graphical interfaces only)

The spacing, in vertical grid units, between the vertical grid lines of the frame.

Data type: INTEGER

Access: Readable/Writeable

Applies to: FRAME widget

If the value is 1, each vertical grid unit is intersected by a line of horizontal grid points. If the value is greater than 1, every *n*th vertical grid unit is intersected by a horizontal line of grid points, where *n* is the value of GRID-FACTOR-VERTICAL. The default value is 6.

The height of a vertical grid unit is defined by the GRID-UNIT-HEIGHT-CHARS or GRID-UNIT-HEIGHT-PIXELS attribute.

Note: Setting this attribute to 1 has the same effect as setting the GRID-FACTOR-HORIZONTAL attribute to 1, because either setting makes all grid points visible in the frame.

GRID-SNAP attribute
(Graphical interfaces only)

Indicates whether widgets should snap to the grid when they are moved or resized.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget

If the GRID-SNAP attribute is TRUE, when widgets are moved or resized they align with (snap to) the closest grid points in the frame. This alignment occurs whether or not the grid points are visible (determined by the GRID-VISIBLE attribute).

The distance between grid points (vertical and horizontal grid units) is defined by the GRID-UNIT-HEIGHT-CHARS, GRID-UNIT-HEIGHT-PIXELS, GRID-UNIT-WIDTH-CHARS, and GRID-UNIT-WIDTH-PIXELS attributes.

GRID-UNIT-HEIGHT-CHARS attribute
(Graphical interfaces only)

The height, in character units, of a vertical grid unit on the frame.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: FRAME widget

This attribute specifies the distance between vertical grid points in the frame. When a widget is moved or resized, it snaps to these grid points within the frame when the GRID-SNAP attribute is set to TRUE. The default value depends on the display resolution and the size of the default system font.

GRID-UNIT-HEIGHT-PIXELS attribute
(Graphical interfaces only)

The height, in pixels, of a vertical grid unit on the frame.

Data type: INTEGER

Access: Readable/Writeable

Applies to: FRAME widget

This attribute specifies the distance between vertical grid points in the frame. When a widget is moved or resized, it snaps to these grid points within the frame when the GRID-SNAP attribute is set to TRUE. The default value is 6.

GRID-UNIT-WIDTH-CHARS attribute
(Graphical interfaces only)

The width, in character units, of a horizontal grid unit on the frame.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: FRAME widget

This attribute specifies the distance between horizontal grid points in the frame. When a widget is moved or resized, it snaps to these grid points within the frame when the GRID-SNAP attribute is set to TRUE. The default value depends on the display resolution and the size of the default system font.

GRID-UNIT-WIDTH-PIXELS attribute
(Graphical interfaces only)

The width, in pixels, of a horizontal grid unit on the frame.

Data type: INTEGER

Access: Readable/Writeable

Applies to: FRAME widget

This attribute specifies the distance between horizontal grid points in the frame. When a widget is moved or resized, it snaps to these grid points within the frame when the GRID-SNAP attribute is set to TRUE. The default value is 6.

GRID-VISIBLE attribute
(Graphical interfaces only)

Indicates whether the grid of a frame is visible.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget

When visible, the grid is a set of points laid out in vertical and horizontal lines. The distance between grid points (vertical and horizontal grid units), whether visible or invisible, is defined by using the GRID-UNIT-HEIGHT-CHARS, GRID-UNIT-HEIGHT-PIXELS, GRID-UNIT-WIDTH-CHARS, and GRID-UNIT-WIDTH-PIXELS attributes. What grid points are visible is determined by the GRID-FACTOR-VERTICAL and GRID-FACTOR-HORIZONTAL attributes, which define the spacing between the visible vertical and horizontal lines of grid points.

When used with scrollable frames, some grid points might not be visible.

GROUP-BOX attribute
(Graphical interfaces only)

Indicates the rectangle is a group box, which surrounds one or more user interface widgets to visually indicate a relationship among the widgets.

Data type: LOGICAL

Access: Readable/Writable

Applies to: RECTANGLE widget

The appearance of a group box changes automatically to conform to the current display setting. When the display is set to the Windows XP Theme, the group box rectangle appears with rounded corners and a single-line border. When the display is set to the Windows Classic Theme, the group box appears with square corners and a shaded double-line border.

If you set this attribute after the widget is realized, its affect is immediate (if the widget is visible).

When TRUE, the ROUNDED attribute setting is ignored.

See also

ROUNDED attribute

HANDLE attribute

A handle to the object.

Data type: HANDLE

Access: Read-only

Applies to: Asynchronous request object handle, AUDIT-CONTROL system handle, AUDIT-POLICY system handle, BROWSE widget (browse, column, and cell), Buffer object handle, Buffer-field object handle, BUTTON widget, Call object handle, Client-principal object handle, CLIPBOARD system handle, CODEBASE-LOCATOR system handle, COLOR-TABLE system handle, COMBO-BOX widget, COMPILER system handle, Data-relation object handle, Data-source object handle, DEBUGGER system handle, DIALOG-BOX widget, DSLOG-MANAGER system handle, EDITOR widget, ERROR-STATUS system handle, FIELD-GROUP widget, FILE-INFO system handle, FILL-IN widget, FONT-TABLE system handle, FRAME widget, IMAGE widget, LAST-EVENT system handle, LITERAL widget, LOG-MANAGER system handle, MENU widget, MENU-ITEM widget, Procedure object handle, ProDataSet object handle, PROFILER system handle, Query object handle, RADIO-SET widget, RCODE-INFO handle, RECTANGLE widget, SAX-attributes object handle, SAX-reader object handle, SAX-writer object handle, SECURITY-POLICY system handle, SELECTION-LIST widget, Server object handle, Server socket object handle, SESSION system handle, SLIDER widget, SOAP-fault object handle, SOAP-fault-detail object handle, SOAP-header object handle, SOAP-header-entryref object handle, Socket object handle, Stream object handle, SUB-MENU widget, Temp-table object handle, TEXT widget, TOGGLE-BOX widget, Transaction object handle, WEB-CONTEXT system handle, WINDOW widget, X-document object handle, X-noderef object handle

You can store this value in a HANDLE variable. You can also use it to associate one widget with another widget or with a system handle. For example, you can assign the HANDLE value of the menu bar to the MENU-BAR attribute of a window, or you can make the window the current window by assigning its HANDLE value to the CURRENT-WINDOW handle.

For query objects, the HANDLE attribute lets you acquire a query object for a static query, as shown in the following code:

```
my-query-handle = QUERY q:HANDLE.
```

The following code uses the HANDLE attribute of a buffer field to retrieve the buffer field's handle:

```
my-buffer-field = city:HANDLE IN BUFFER Customer.
```

The preceding code requires that you know the name of the field (in this case, `"City"`) at compile time. The following code, which performs the same task, does not require the name of the field:

```
my-buffer = BUFFER Customer:HANDLE
my-buffer-field = my-buffer:BUFFER-FIELD("City").
```

The value of the HANDLE attribute is guaranteed to be unique among the HANDLE attributes for all object instances in an ABL session.

HANDLER attribute

A handle to the procedure containing the SAX callbacks.

Note: To get the handle to an object-oriented ABL object containing methods as callback for SAX events use the EVENT-HANDLER-OBJECT attribute.

Data type: HANDLE

Access: Readable/Writable

Applies to: SAX-reader object handle

The default is a handle to the procedure that contains the SAX-PARSE( ) method, or the SAX-PARSE-FIRST( ) and SAX-PARSE-NEXT( ) methods.

When SAX-PARSE( ), SAX-PARSE-FIRST( ), or SAX-PARSE-NEXT( ) executes, the SAX-reader object looks for callbacks only in the procedure or object whose handle is stored in the HANDLER. HANDLER must be a valid procedure or object handle and cannot be a proxy.

Note: It is permissible for both the driver procedure (the procedure that contains the SAX-PARSE( ) method, or the SAX-PARSE-FIRST( ) and SAX-PARSE-NEXT( ) methods) and callback handler procedure or object of a SAX application to reside on a remote application server. If this occurs, callbacks are invoked local to the application server.

Callbacks can reside within a special handler procedure file that is run persistently or within the driver procedure.

Within a procedure file, to get a handle to the procedure file, use the THIS-PROCEDURE handle. The following fragment assigns HANDLER a handle to the current procedure:

```
hSaxReader:HANDLER = THIS-PROCEDURE.
```

Notes

- Assigning a value other than **?** to the HANDLER attribute will cause an update to the EVENT-HANDLER-OBJECT attribute changing it to the unknown value.
- Assigning a value other than **?** to the EVENT-HANDLER-OBJECT attribute will cause an update to the HANDLER attribute changing it to the unknown value.

HAS-LOBS attribute

Returns TRUE if the Buffer object has BLOB or CLOB fields defined in it. Otherwise, it returns FALSE.

Data type: LOGICAL

Access: Read-only

Applies to: Buffer object handle

HAS-RECORDS attribute

This attribute returns TRUE when the corresponding temp-table has records. It returns FALSE when the temp-table does not have any records, or the temp-table is in an UNPREPARED state.

Data type: LOGICAL

Access: Read-only

Applies to: Temp-table object handle

Height property
(Windows only; Graphical interfaces only)

The height of the control-frame and control-frame COM object, in pixels.

Return type: INTEGER

Access: Readable/Writeable

Applies to: CONTROL-FRAME widget, COM object

Setting this value changes the HEIGHT-CHARS attribute and HEIGHT-PIXELS attribute of the corresponding control-frame widget to an equivalent value.

Note: References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the Accessing COM object properties and methods.

HEIGHT-CHARS attribute

The height, in character units, of the widget. The HEIGHT-CHARS attribute of the SESSION handle contains the height of the display.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: BROWSE widget (browse and cell), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SESSION system handle, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

For combo boxes, field groups, and the SESSION handle, this attribute is read-only.

In character mode, the HEIGHT-CHARS attribute must be set before the browse is realized. Attempting to set the attribute after realization of the browse causes a run-time error.

For control-frames, the HEIGHT-CHARS attribute maps to the Height property of the control-frame COM object (ActiveX control container).

For browses, the HEIGHT-CHARS attribute sets the decimal height, in characters, of the browse without changing the height of the browse's rows. If you change the value of a browse's HEIGHT-CHARS or HEIGHT-PIXELS attribute, the number of rows that appear in the viewport might change.

HEIGHT-PIXELS attribute

The height, in pixels, of the widget. The HEIGHT-PIXEL attribute of the SESSION handle contains the height of the display.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse and cell), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SESSION system handle, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

For combo boxes, field groups, and the SESSION handle, this attribute is read-only.

In character mode, the HEIGHT-PIXELS attribute must be set before the browse is realized. Attempting to set the attribute after realization of the browse causes a run-time error.

For control-frames, the HEIGHT-PIXELS attribute maps to the Height property of the control-frame COM object (ActiveX control container).

For browses, the HEIGHT-PIXELS attribute sets the decimal height, in pixels, of the browse without changing the height of the browse's rows. If you change the value of a browse's HEIGHT-CHARS or HEIGHT-PIXELS attribute, the number of rows that appear in the viewport might change.

HELP attribute

The help text for a field.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), Buffer-field object handle, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

For this attribute to have effect, the window that contains the specified widget must have its STATUS-AREA attribute set to TRUE. The text stored in the HELP attribute displays in the status area of the containing window when the widget has input focus. The HELP attribute text overrides any status-area text issued by the STATUS statement.

HIDDEN attribute

Indicates whether to "hide" a widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

Setting the HIDDEN attribute to TRUE prevents the widget from being displayed implicitly. For a field-level widget, child frame, or child window, this means that the widget is not automatically made visible when the containing frame or parent window becomes visible. The widget does not appear unless one of the following occurs:

- It is forced to receive user input (for example, using a SET or PAUSE statement).
- It is explicitly displayed using a VIEW statement or by setting its VISIBLE attribute to TRUE.

Any action that explicitly displays the widget also resets the HIDDEN attribute to FALSE. If the widget is already visible, setting its HIDDEN attribute to TRUE makes that widget and any widgets it parents (and their descendants) invisible (VISIBLE is set to FALSE). The default value of the HIDDEN attribute is FALSE for all widgets.

In windows, setting the HIDDEN attribute to TRUE prevents implicit display of the hidden window when you:

- Invoke DISPLAY, ENABLE, and VIEW statements for frames of the window
- View an ancestor or descendant window of the hidden window

This limits flashing side effects caused during set up of the application user interface. In windows, this attribute is not supported in character mode.

For frames and dialog boxes, setting the HIDDEN attribute to TRUE prevents implicit display of the frame or dialog box when you invoke DISPLAY or ENABLE statements for the widget or its descendant frames. This allows the frame or dialog box to remain invisible during actions that set it up. The HIDDEN attribute has no effect on DISPLAY statements directed to a file, pipe, or printer.

Note: Setting a frame or field-level widget's VISIBLE attribute to TRUE also displays any parent or ancestor frames, even if their HIDDEN attributes are set to TRUE (resetting the HIDDEN attributes, if necessary). However, setting a window's VISIBLE attribute to TRUE only displays the window if there are no ancestor windows with their HIDDEN attribute set to TRUE. In any case, the window's own HIDDEN attribute is set to FALSE.

For field-level widgets and frames parented by other frames, setting the HIDDEN attribute to TRUE prevents implicit display of the field-level widget or child frame when its containing frame or dialog box is displayed. If the frame or dialog box containing the widget is visible, setting HIDDEN to FALSE for the widget makes the widget visible (the VISIBLE attribute is set to TRUE). If the containing frame or dialog box is not visible, setting HIDDEN to FALSE has no effect on the VISIBLE attribute of the widget.

Note: The HIDE statement sets the VISIBLE attribute for the widget to FALSE. It only sets the HIDDEN attribute to TRUE if you hide a field-level widget or child frame whose containing frame is still visible.

HonorProKeys property
(Windows only; Graphical interfaces only)

Determines who processes the **GO**, **ENDKEY**, **HELP**, and **TAB** keys: the AVM, or the ActiveX control to which the property applies.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Any ActiveX control

If the property is TRUE, which is the default, the AVM intercepts these keys and processes them as normal ABL key events. If the property is FALSE, the keystrokes are sent to the ActiveX control for processing.

Note: This property resembles the HonorReturnKey property, which governs processing of the **RETURN** key, but whereas the default setting for HonorProKeys is TRUE (the AVM gets the event), the default setting for HonorReturnKey is FALSE (the ActiveX control gets the event).

HonorReturnKey property
(Windows only; Graphical interfaces only)

Determines who processes the **RETURN** key: the AVM, or the ActiveX control to which the property applies.

If the property is TRUE, the AVM intercepts the key and processes it as a normal ABL **RETURN** key event. If the property is FALSE, which is the default, the keystroke is sent to the ActiveX control for processing.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Any ActiveX control

If a frame has a default button and an ActiveX control, and you want the **RETURN** key to activate the default button regardless of who has focus, you must set the HonorReturnKey property of the ActiveX control to TRUE. Similarly, if a frame has a default button and several ActiveX controls, and you want the **RETURN** key to activate the default button regardless of who has focus, you must set the HonorReturnKey property of all the ActiveX controls in the frame to TRUE.

This property resembles the HonorProKeys property, which governs processing of several other keys, but whereas the default setting for HonorReturnKey is FALSE (the ActiveX control gets the event), the default setting for HonorProKeys is TRUE (the AVM gets the event).

HORIZONTAL attribute

The orientation of a slider, or of radio buttons in a radio set.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: RADIO-SET widget, SLIDER widget

If HORIZONTAL is TRUE, the orientation is horizontal; if it is FALSE, the orientation is VERTICAL. By default, the orientation of sliders is horizontal and the orientation of radio sets is vertical.

You can set this attribute only before the widget is realized.

HTML-CHARSET attribute
(WebSpeed only)

The ABL version (as opposed to the MIME version) of the code page name of a Web request. Set by ABL when a WebSpeed application incorporates dynamic code page support. The default is blank.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

Once the AVM sets HTML-CHARSET, each time a Web browser sends a Web request to the application:

1. The request is received by the WebSpeed Agent. The AVM converts it from the HTML-CHARSET code page to the Agent's CPINTERNAL code page.
2. The Agent responds to the request with a Web page. The AVM converts the Web page from the Agent's CPINTERNAL code page to the HTML-CHARSET code page.

Caution: If the application modifies the contents of HTML-CHARSET, dynamic code-page support might fail.

HTML-END-OF-LINE attribute

Defaults to the newline character (ASCII 10; '~n'; '\n'). A null string value causes a NEWLINE character (not a null string) to be output. You might want to set this to "~n" (the NEWLINE character) or to the null string (to force the NEWLINE character). Depending on the other attribute values, using the NEWLINE rather than the <BR> tag can result in more readable output when viewing document source in a browser.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HTML-END-OF-PAGE attribute

Between stream pages, defaults to "<HR>". Output between stream pages to visually break up the sectioning caused by the PAGED or PAGE-SIZE options of the OUTPUT TO "WEB" statement. Does not affect the line count of any stream page.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HTML-FRAME-BEGIN attribute

Before a SpeedScript frame, defaults to "<PRE>". Generally, if you change this value you must change the value of HTML-FRAME-END. Output only before the data row(s) for the current iteration of a DOWN frame, not to column headers (see also HTML-HEADER-BEGIN and HTML-HEADER-END). Applies to any side-labels displayed in the frame, whether or not the frame is a DOWN frame.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HTML-FRAME-END attribute

After a SpeedScript frame, defaults to "</PRE>". Generally, if you change this value you must change the value of HTML-FRAME-BEGIN. Output at the end of the data row(s) for the current iteration of a DOWN frame.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HTML-HEADER-BEGIN attribute

Before the column headers of a SpeedScript frame, defaults to "<PRE>". Generally, if you change this value you must change the value of HTML-HEADER-END. Output at the beginning of the column header section of a DOWN frame.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HTML-HEADER-END attribute

After the column headers of a SpeedScript frame, defaults to "</PRE>". Generally, if you change this value you must change the value of HTML-HEADER-BEGIN. Output at the end of the column header section of a DOWN frame.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HTML-TITLE-BEGIN attribute

Before a SpeedScript frame title, Defaults to the null string (""), no text. Generally, if you change this value you must change the value of HTML-TITLE-END. Output before the frame's TITLE value. Setting to a color or bold tag might improve readability.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HTML-TITLE-END attribute

After a SpeedScript frame title, defaults to the null string (""), no text. Generally, if you change this value you must change the value of HTML-TITLE-BEGIN. Output after the frame's TITLE value.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HWND attribute
(Windows only; Graphical interfaces only)

An integer value for a Windows handle to the window that contains the widget.

Data type: INTEGER

Access: Read-only

Applies to: BROWSE widget (browse and cell), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

This attribute is supported for dynamic link library (DLL) access only in Windows. Some DLL routines require that you pass this value.

For ABL window widgets, the Windows window that contains the widget is actually the parent of the Windows widget referenced by HWND. Thus, to obtain the handle of the Windows window that contains the ABL window, you must pass the value of HWND to the `GetParent( )` function (in the `user32.dll`). Pass the result of `GetParent( )` to the DLL routine that requires it.

ICFPARAMETER attribute

A character string that supplies Internet Component Framework (ICF) procedures (in Progress Dynamics) with ICF-related data.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

Use the ICF Parameter (-icfparam) parameter to specify a character string, at the start of an ABL session, that can be accessed from ABL procedures within the ICF framework.

Note: The ICF Parameter (-icfparam) parameter is reserved for use by the ICF and procedures that have been integrated with the ICF. Using this parameter for any purpose other than operating within the ICF framework will interfere with your ability to integrate your application with that framework at a later time.

ICON attribute

Returns the name of the icon loaded by LOAD-ICON( ).

Data type: CHARACTER

Access: Read-only

Applies to: WINDOW widget

IGNORE-CURRENT-MODIFIED attribute

This attribute is supported only for backward compatibility. Use the PREFER-DATASET attribute instead.

IMAGE attribute

Returns the name of the image loaded by LOAD-IMAGE( ).

Data type: CHARACTER

Access: Read-only

Applies to: BUTTON widget, IMAGE widget

IMAGE-DOWN attribute

Returns the name of the image loaded by LOAD-IMAGE-DOWN( ).

Data type: CHARACTER

Access: Read-only

Applies to: BUTTON widget

IMAGE-INSENSITIVE attribute

Returns the name of the image loaded by LOAD-IMAGE-INSENSITIVE( ).

Data type: CHARACTER

Access: Read-only

Applies to: BUTTON widget

IMAGE-UP attribute

Returns the name of the image loaded by LOAD-IMAGE( ) or LOAD-IMAGE-UP( ).

Data type: CHARACTER

Access: Read-only

Applies to: BUTTON widget

IMMEDIATE-DISPLAY attribute
(Graphical interfaces only)

The frequency of screen updates for the current session.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

If TRUE, the AVM updates the display for every I/O operation, including DISPLAY statements. If FALSE, the AVM does not update the display until a statement blocks for input, such as an UPDATE statement. FALSE is the default setting. A TRUE setting provides more accurate screen displays during long display loops at the price of slower performance.

In Windows, this attribute has a similar effect on interactive I/O to setting the MULTITASKING-INTERVAL attribute to a low non-zero value, providing a more frequent display refresh. This attribute also provides the same functionality as the ImmediateDisplay parameter in the current environment (which might be the Registry (Windows only) or an initialization file). For more information on environments, see the topics on user interface environments in *Manage ABL Applications*.

IMPORT-NODE( ) method

Import a copy of a node from another document into this document. The first parameter must be a valid X-noderef handle and will refer to the newly copied XML node if the method succeeds. The new node is associated with this document, but must be appended or inserted with APPEND-CHILD( ) or INSERT-BEFORE( ) to become part of the structure.

Return type: LOGICAL

Applies to: X-document object handle

Syntax

```
IMPORT-NODE ( x-node , x-source-node , deep )
```

x-node

A valid X-noderef handle to use for the new XML node.

x-source-node

A valid X-noderef handle that represents the node to import from.

deep

A logical that if TRUE specifies that the whole sub-tree is to be copied. The default value is FALSE.

If hDoc is an existing and loaded X-document and hDocCopy is existing but empty and hRoot and hRootCopy are X-noderefs, you can copy hDoc to hDocCopy as follows:

```
/* Associates hRoot with the root node of the hDoc document. */

hDoc:GET-DOCUMENT-ELEMENT(hRoot).
hDocCopy:IMPORT-NODE(hRootCopy, hRoot, TRUE).
hDocCopy:APPEND-CHILD(hRootCopy).
```

IMPORT-PRINCIPAL( ) method

Imports an unsealed or sealed security token as a RAW data type, creates a corresponding client-principal object, and assigns its handle value to the current Client-principal object handle. This corresponding client-principal object assumes all the property and attribute settings of the source client-principal object when it was exported using the EXPORT-PRINCIPAL( ) method.

Return type: LOGICAL

Applies to: Client-principal object handle

Syntax

```
IMPORT-PRINCIPAL ( expression )
```

expression

A RAW expression containing the security token to import. If the specified expression has the Unknown value (?), the AVM raises a run-time error.

Caution: To ensure that the transported identity is a trusted identity, it is your responsibility to ensure that you obtain the security token you are importing from a secure source and in a secure manner.

If you set properties on the receiving client-principal object handle before calling this method, either by having previously imported a security token or by setting them in a client-principal that you have created, the property values are lost.

If successful, this method returns TRUE. Otherwise, it returns FALSE with any messages returned in the ERROR-STATUS system handle.

This method also checks the LOGIN-EXPIRATION-TIMESTAMP attribute. If the client-principal object has expired before you can import it, the AVM sets the LOGIN-STATE attribute to `"EXPIRED"` and the method returns TRUE.

Depending on its sealed or unsealed state, you can use the imported object to assert or set the user identity it represents using the SEAL( ) method, SECURITY-POLICY:SET-CLIENT( ) method, or SET-DB-CLIENT function, as appropriate.

Calling this method does not generate an audit event or an audit record.

See also

EXPORT-PRINCIPAL( ) method, SET-CLIENT( ) method, SET-DB-CLIENT function

INCREMENT-EXCLUSIVE-ID( ) method

Gets the amount by which to increment the exclusive ID of a Web request for state-aware agents. Do not access this method.

Return type: INTEGER

Applies to: WEB-CONTEXT system handle

INDEX attribute

The subscript value of the array element referenced by the current widget.

Data type: INTEGER

Access: Read-only

Applies to: BROWSE widget (column and cell), COMBO-BOX widget, FILL-IN widget, TEXT widget, TOGGLE-BOX widget

If the widget references a field or variable with no extents (that is, not as an array element), this attribute returns 0.

INDEX-INFORMATION attribute

A character string consisting of a comma-separated list of the index or indexes the query uses at the level of join specified.

Data type: CHARACTER

Access: Read-only

Applies to: Query object handle

Syntax

```
INDEX-INFORMATION ( n )
```

n

An integer expression that evaluates to the level of join for which you want index information.

If the index or indexes do not have bracketing, the first entry in the list is the CHARACTER string "WHOLE-INDEX," and the second entry in the list is name of the index.

Before you use INDEX-INFORMATION on a dynamic query, you must prepare the query using the QUERY-PREPARE method.

Before you can use the INDEX-INFORMATION attribute on a static query, you must define the query using the DEFINE QUERY statement's RCODE-INFORMATION option.

The following example prints out the PREPARE-STRING, analyzes the INDEX-INFORMATION, and prints a list of bracketed and whole-index indexes:

r-iinfo.p

```
DEFINE VARIABLE hQuery AS HANDLE  NO-UNDO.
DEFINE VARIABLE ix     AS INTEGER NO-UNDO.
DEFINE VARIABLE jx     AS INTEGER NO-UNDO.

DEFINE QUERY q FOR Customer, Order, OrderLine SCROLLING.

hQuery = QUERY q:HANDLE.
hQuery:QUERY-PREPARE("FOR EACH Customer WHERE Customer.CustNum < 3,
 EACH Order OF Customer, EACH OrderLine").
hQuery:QUERY-OPEN.

MESSAGE "prepare string is" hQuery:PREPARE-STRING.

REPEAT ix = 1 TO hQuery:NUM-BUFFERS:
 jx = LOOKUP("WHOLE-INDEX", hQuery:INDEX-INFORMATION(ix)).
 IF jx > 0 THEN
   MESSAGE "inefficient index" ENTRY(jx + 1, hQuery:INDEX-INFORMATION(ix)).
 ELSE
   MESSAGE "bracketed index use of" hQuery:INDEX-INFORMATION(ix).
END.
```

INDEX-INFORMATION( ) method

Returns index information in a comma-separated list for the *i*th index in the buffer's table.

The returned comma-separated list consists of the following in the specified order:

- The index name
- Three integer values of value 0 (FALSE) or 1 (TRUE) depending on whether the index is unique, primary or a word index
- The names of the index fields, each followed by a 0 (ascending) or 1 (descending)

Return type: CHARACTER

Applies to: Buffer object handle

Syntax

```
INDEX-INFORMATION ( i )
```

i

The relative number of the buffer table's index for which you want information.

When the index argument, *i*, is beyond the number of indices in the table or is otherwise invalid, the Unknown value (?) is returned.

The following code fragment requests information about the third index in the customer table:

```
buffCustHdl = BUFFER Customer:HANDLE.
IndexVar = buffCustHdl:INDEX-INFORMATION(3).
```

The returned string would look like: "CustNum,1,1,0,CustNum,0" which means that the third index in the Customer table is called "CustNum". It is unique and primary, and is not a word index and it consists of one ascending component, CustNum.

IN-HANDLE attribute

Holds one of the following:

- A handle to a persistent procedure (a procedure invoked with the PERSISTENT attribute set to TRUE or PROCEDURE-TYPE attribute set to "PERSISTENT")
- A handle to a single-run or singleton procedure (a procedure invoked with the PROCEDURE-TYPE set to "SINGLE-RUN" or "SINGLETON", respectively)
- A handle to an object, such as a buffer or socket, which has attributes you want to get or set dynamically or methods you want to run dynamically

Data type: HANDLE

Access: Readable/Writable

Applies to: Call object handle

You can set IN-HANDLE using the following syntax:

Syntax

```
call-object:IN-HANDLE [ = {handle-expression | char-expression} ]
```

handle-expression

A HANDLE expression.

char-expression

A CHARACTER expression indicating the name of a system object, such as "SESSION" or "FILE-INFO", whose handle is returned.

The default is the Unknown value (?).

IN-HANDLE will be set automatically to a procedure handle when you run a persistent, single-run, or singleton procedure by calling the INVOKE( ) method on the call object:

```
call-object:INVOKE( )
```

The following table lists the different ways of setting IN-HANDLE.

Table 82: Setting IN-HANDLE

If you want to access. . .	Then IN-HANDLE. . .
A procedure as persistent, single-run, or singleton	Is automatically set to the procedure handle after calling the INVOKE( ) method on the call object
An internal procedure or user-defined function of a previously invoked persistent, single-run, or singleton procedure	Must be manually set to the handle of the existing object, unless IN-HANDLE already holds the procedure handle from a previous INVOKE( )
Any handle-based object whose attributes you want to get or set dynamically	Must be manually set IN-HANDLE to the handle of the existing object, such as a temp-table, buffer, or socket

When you use IN-HANDLE to call an internal procedure, IN-HANDLE affects INVOKE( ) the same way the IN *proc-handle* phrase affects the RUN statement. Similarly, when you use IN-HANDLE to call a user-defined function, it affects INVOKE( ) the same way the IN *proc-handle* phrase affects the DYNAMIC-FUNCTION function. In both cases, IN-HANDLE specifies the instance of the external procedure that contains the internal procedure or user-defined function.

When IN-HANDLE is used to get or set an attribute or to invoke a method, it represents a handle to the object to which the attribute applies. If the attribute applies to a system object such as the SESSION handle or the FILE-INFO handle, IN-HANDLE can be set to a character string such as "SESSION" or "FILE-INFO" that indicates the name of the system object.

Note: You can run a procedure as persistent, single-run, or singleton statically or dynamically. Similarly, you can run any of the persistent, single-run, or singleton procedure's internal procedures and user-defined functions statically or dynamically.

For information on dynamically invoking logic that resides on an application server, see the reference entry for SERVER attribute.

INHERIT-BGCOLOR attribute
(Graphical interfaces only)

Indicates whether field-level widgets inherit the background color of their containing dialog-box or frame widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget, SESSION system handle

The default value is TRUE. That is, field-level widgets inherit the background color of the containing dialog-box or frame widget, by default.

When set to FALSE for a dialog-box or frame widget, the following field-level widgets will not inherit the background color of their containing widget: BROWSE, COMBO-BOX (all types), EDITOR, FILL-IN (NATIVE and Enabled), and SELECTION-LIST.

You must set this attribute before realizing the dialog-box or frame widgets.

For static dialog-box and frame widgets, you can also change the default background color inheritance on a frame-by-frame basis by specifying the INHERIT-BGCOLOR option in the Frame phrase when you define the widget.

You can also set this attribute to FALSE on the SESSION system handle to change the default background color inheritance for all dialog-box and frame widgets in the session. Setting this attribute affects dialog-box and frame widgets created only after the attribute is set; it does not affect existing widgets.

Note: If you set this attribute on the SESSION system handle in a procedure that also contains static dialog-box or frame widget definitions, the static widgets will not be affected by the attribute setting because they are created before the procedure begins executing. For this reason, Progress Software Corporation recommends that you set this attribute on the SESSION system handle in an initialization procedure that does not contain static widget definitions.

Setting the BGCOLOR attribute for a field-level widget within the dialog-box or frame overrides any INHERIT-BGCOLOR attribute or option settings.

See also

BGCOLOR attribute, Frame phrase, INHERIT-FGCOLOR attribute

INHERIT-FGCOLOR attribute
(Graphical interfaces only)

Indicates whether field-level widgets inherit the foreground color of their containing dialog-box or frame widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget, SESSION system handle

The default value is TRUE. That is, field-level widgets inherit the foreground color of the containing dialog-box or frame widget, by default.

When set to FALSE for a dialog-box or frame widget, the following field-level widgets will not inherit the foreground color of their containing widget: BROWSE, COMBO-BOX (all types), EDITOR, FILL-IN (NATIVE and Enabled), and SELECTION-LIST.

You must set this attribute before realizing the dialog-box or frame widgets.

For static dialog-box and frame widgets, you can also change the default foreground color inheritance on a frame-by-frame basis by specifying the INHERIT-FGCOLOR option in the Frame phrase when you define the widget.

You can also set this attribute to FALSE on the SESSION system handle to change the default foreground color inheritance for all dialog-box and frame widgets in the session. Setting this attribute affects dialog-box and frame widgets created only after the attribute is set; it does not affect existing widgets.

Note: If you set this attribute on the SESSION system handle in a procedure that also contains static dialog-box or frame widget definitions, the static widgets will not be affected by the attribute setting because they are created before the procedure begins executing. For this reason, Progress Software Corporation recommends that you set this attribute on the SESSION system handle in an initialization procedure that does not contain static widget definitions.

Setting the FGCOLOR attribute for a field-level widget within the dialog-box or frame overrides any INHERIT-FGCOLOR attribute or option settings.

See also

FGCOLOR attribute, Frame phrase, INHERIT-BGCOLOR attribute

INITIAL attribute

The value of the INITIAL schema field (which is always CHARACTER). If the INITIAL schema field has the Unknown value (?), the value of the INITIAL attribute is the null string ("").

Data type: CHARACTER

Access: Read-only

Applies to: Buffer-field object handle

INITIAL renders its dates and numeric values in LOCAL format, subject to the DATE-FORMAT and NUMERIC-FORMAT attributes of the SESSION system handle.

See also

DEFAULT-STRING attribute, DEFAULT-VALUE attribute

INITIALIZE( ) method

Simplifies initialization of client-principal object attributes that are required and commonly used to assert a given identity for user authentication. This method can be called on any client-principal object, whether it is sealed or unsealed. Each time the method is called, it returns the object to the INITIAL state (see the LOGIN-STATE attribute entry) initialized with values from the parameter list.

Return type: LOGICAL

Applies to: Client-principal object handle

Syntax

```
INITIALIZE ( qualified-user-id [ , session-id [ , expiration
                               [ , primary-passphrase ] ] ] )
```

qualified-user-id

A character expression that evaluates to a fully qualified user ID (user name and domain name delimited by the '@' character), which is equivalent to setting the USER-ID attribute and the DOMAIN-NAME attribute on the object. For more information on valid values, see the reference entry for this attribute.

session-id

An optional character expression that evaluates to the user's application login session ID. This value sets the SESSION-ID attribute on the object. If not specified or the parameter evaluates to the Unknown value (?), OpenEdge generates a 22-character Base64-encoded UUID value that is suitable for use in OpenEdge auditing.

expiration

An optional DATETIME-TZ expression that evaluates to a date and time value that specifies the expiration of the client-principal user credentials. A valid value sets the LOGIN-EXPIRATION-TIMESTAMP attribute on the object. If not specified or the parameter evaluates to the Unknown value (?), no expiration is set for the client-principal.

primary-passphrase

An optional character expression that evaluates to the cleartext or encrypted value of the user's account password. For information on setting an encrypted password value, see ENCRYPT-AUDIT-MAC-KEY( ) method and the topics on encrypted passwords in *OpenEdge Programming Interfaces*.

This value sets the PRIMARY-PASSPHRASE attribute on the object, which is used in any OpenEdge-performed user authentication operation. If not specified or the parameter evaluates to the Unknown value (?), the PRIMARY-PASSPHRASE attribute is not set.

Note:

This value has no relation to the domain access code used to seal a client-principal object.

OpenEdge does not store the *primary-passphrase* value. Once the client-principal object is sealed, OpenEdge removes all trace of this value from the client-principal.

Use an encrypted *primary-passphrase* value especially when you export the unsealed client-principal to a remote authentication service, or to any other ABL session prior to sealing the object. For more information on exporting a client-principal, see EXPORT-PRINCIPAL( ) method.

You can change these (and other writeable) attribute settings at any time after invoking this method until the client-principal object is sealed.

ABL raises ERROR if:

- The *qualified-user-id* parameter evaluates to the Unknown value (?)
- The *expiration* parameter does not evaluate to a valid DATETIME-TZ value
- The *session-id* parameter evaluates to a blank character string

Note: If an error occurs when invoking this method, the existing client-principal object remains unchanged.

See also

PRIMARY-PASSPHRASE attribute, QUALIFIED-USER-ID attribute, SESSION-ID attribute

INITIALIZE-DOCUMENT-TYPE( ) method

Creates a new XML document, initializes the document based on the referenced DTD, and creates its root node.

Return type: LOGICAL

Applies to: X-document object handle

Syntax

```
INITIALIZE-DOCUMENT-TYPE ( namespace-uri , root-node-name , public-id ,
  system-id )
```

namespace-uri

A character expression representing the namespace Uniform Resource Identifier (URI) you want associated with the root node of the XML document. The *namespace-uri* must be unique and persistent.

root-node-name

A character expression representing the name of the root node as defined in the XML document. If you are using namespaces and you want to associate a prefix with the namespace, you must qualify this node name with the *namespace-uri* and a colon character prefix (for example, *namespace-uri:root-node-name*). You must explicitly set the xmlns attribute on the root node.

public-id

An optional character expression representing the public ID of the DTD. Currently, there is no way to retrieve a DTD based on a public ID.

system-id

A required character expression representing the system ID of the DTD. This contains the path to the DTD which is either a file system path or an HTTP URL. The ABL parser uses this information to retrieve the DTD when parsing the document.

The following example initializes an X-DOCUMENT with a DTD reference and adds the proper namespace declaration, if the namespace URI is not empty:

```
DEFINE INPUT  PARAMETER namespaceURI AS CHARACTER NO-UNDO.
DEFINE INPUT  PARAMETER rootNodeName AS CHARACTER NO-UNDO.
DEFINE INPUT  PARAMETER publicId     AS CHARACTER NO-UNDO.
DEFINE INPUT  PARAMETER systemId     AS CHARACTER NO-UNDO.
DEFINE OUTPUT PARAMETER hDocument    AS HANDLE    NO-UNDO.

DEFINE VARIABLE hNsDecl   AS HANDLE  NO-UNDO.
DEFINE VARIABLE hRootNode AS HANDLE  NO-UNDO.
DEFINE VARIABLE errStat   AS LOGICAL NO-UNDO.
DEFINE VARIABLE found     AS INTEGER NO-UNDO.

/* Create X-DOCUMENT and intialize it. */
CREATE X-DOCUMENT hDocument.
errStat = hDocument:INITIALIZE-DOCUMENT-TYPE
  (namespaceURI, rootNodeName, publicId, systemId).
IF NOT errStat THEN DO:
  DELETE OBJECT hDocument.
  LEAVE.
END.

/* If using namespaces, create X-NODEREF for namespace declaration. */
IF LENGTH(namespaceURI) > 0 THEN DO:
  CREATE X-NODEREF hNsDecl.
  CREATE X-NODEREF hRootNode.

  /* Look for a colonized name in rootNodeName. */
  found = INDEX(rootNodeName, ":").
 IF found > 0 THEN DO:
  /* Namespace declarations are special kinds of attributes that belong in
      the http://www.w3.org/2000/xmlns/ namespace.*/
  errStat = hDocument:CREATE-NODE-NAMESPACE(hNsDecl,
   "http://www.w3.org/2000/xmlns/", "xmlns:" +
   SUBSTRING(rootNodeName, 1, found - 1), "attribute").
 END.
 ELSE DO:
    /* Use the default namespace, which does not need a namespace declaration
       prefix, assign it to the http://www.w3.org/2000/xmlns/ namespace.*/
  errStat = hDocument:CREATE-NODE-NAMESPACE(hNsDecl,
   "http://www.w3.org/2000/xmlns/", "xmlns", "attribute").
 END.
 IF NOT errStat THEN LEAVE.

 /* Set the value of the namespace attribute to the namespace URI. */
 hNsDecl:NODE-VALUE = namespaceURI.

 /* Retrieve the root node and add the namespace declaration to it. */
 errStat = hDocument:GET-DOCUMENT-ELEMENT(hRootNode).
 IF NOT errStat THEN LEAVE.
 errStat = hRootNode:SET-ATTRIBUTE-NODE(hNsDecl).
END.
```

```
/* Free up the temporary X-NODEREFS. */
IF VALID-HANDLE(hNsDecl) THEN
 DELETE OBJECT hNsDecl.
IF VALID-HANDLE(hRootNode) THEN
 DELETE OBJECT hRootNode.

/* If an error occurred, free up the X-DOCUMENT. */
IF NOT errStat THEN
 DELETE OBJECT hDocument.
```

INITIATE( ) method

Initializes the Debugger, but does not pass control to the Debugger immediately. To start the Debugger from the procedure in application mode, you must set a breakpoint using the SET-BREAK( )method that the procedure encounters. When the procedure encounters the breakpoint, the Debugger takes control of the procedure at that point.

Return type: LOGICAL

Applies to: DEBUGGER system handle

Syntax

```
INITIATE ( )
```

If the INITIATE( ) method successfully initializes the Debugger or the Debugger is already initialized, the method returns TRUE. Otherwise, it returns FALSE with no effect.

Note: To use this method, the Application Debugger must be installed.

All other Debugger attributes and methods (except the DEBUG( ) method) have no effect unless you first initialize the Debugger with this method or start the Debugger from the OpenEdge ADE. If the Debugger is already initialized and running (for example, by running ABL with the `-debug` startup parameter), this method has no effect.

INNER-CHARS attribute

The number of data columns within a selection list or editor widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: EDITOR widget, SELECTION-LIST widget

This attribute is more portable than the WIDTH-CHARS attribute.

For editor widgets, this attribute can set the word wrap margin for the WORD-WRAP attribute. For more information, see the WORD-WRAP attribute reference entry.

If a selection list is not already realized and you reference its INNER-CHARS attribute, the AVM realizes the widget.

INNER-LINES attribute

The number of data lines within a combo-box drop down list, editor widget, or selection list.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, SELECTION-LIST widget

For combo boxes, this attribute has meaning only in Windows.

This attribute is more portable than the HEIGHT-CHARS attribute. For combo boxes, the value must be 3 or greater.

If a selection list or combo box is not already realized and you reference its INNER-LINES attribute, the AVM realizes the widget.

For browses, this attribute applies only to combo-box browse columns.

INPUT-VALUE attribute

Used for data-representation widgets, such as field-level widgets that represent variables or database fields. The value for the INPUT-VALUE attribute is the unformatted SCREEN-VALUE of a widget.

Data type: Same as the field or variable associated with the widget

Access: Read-only

Applies to: BROWSE widget (cell), COMBO-BOX widget, EDITOR widget, FILL-IN widget, LITERAL widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

For any widget that has a SCREEN-VALUE, INPUT-VALUE returns the unformatted value of the widget's SCREEN-VALUE, in the native data type of the widget.

When you use the INPUT-VALUE attribute, there are no formatting characters and the data type is the data type of the associated field. The relationship between a field's INPUT-VALUE and the value in the record buffer is the same as the relationship between the SCREEN-VALUE and the record buffer. For example, changing the record buffer does not affect the INPUT-VALUE.

When you use the DISPLAY, PUT, or MESSAGE statements for the INPUT-VALUE of a widget that has the DATE data type, you see the formatted value because these statements automatically format DATE values. To display an unformatted date, first assign INPUT-VALUE to an integer variable, and then display the variable.

See also

SCREEN-VALUE attribute

INSERT( ) method

Inserts a new item before a specified item in a combo box or selection list. The new item can consist of a label, a list of labels, or a label-value pair.

Return type: LOGICAL

Applies to: BROWSE widget (column), COMBO-BOX widget, SELECTION-LIST widget

Syntax

```
INSERT (
  { new-item-list | new-label , new-value }
  , { list-item | list-index }  )
```

new-item-list

A character-string expression that specifies a single item or a delimiter-separated list of items to add to the widget.

new-label

A character-string expression that specifies the label of a label-value pair to add to the widget.

new-value

The new value assigned when a user selects the label.

Note: If the widget's entries consist of single items, use *new-item-list*. If the widget's entries consist of label-value pairs, use *new-label* and *new-value*.

list-item

A character-string expression that specifies a single value in the widget.

list-index

An integer expression that specifies the ordinal position of an existing entry in the widget. The first item is specified by 0 and the last item is specified by -1.

The delimiter is the value of the DELIMITER attribute, which is a comma by default. If the method is successful, it returns TRUE.

For browses, this method applies only to combo-box browse columns.

Note: If the widget's entries consist of single items, each call to INSERT can add multiple entries. If the widget's entries consist of label-value pairs, each call to INSERT can add one entry.

INSERT-ATTRIBUTE( ) method

For a SAX-writer object, adds a single attribute to a start tag in the XML document represented by the SAX-writer object. For a SAX-attributes object, inserts an attribute and its value into the SAX-attributes object.

Caution: The SAX specifications specifically state that the order of attributes in an XML element's list are not guaranteed.

Return type: LOGICAL

Applies to: SAX-attributes object handle, SAX-writer object handle

Syntax

```
INSERT-ATTRIBUTE ( attribute-name , attribute-value [ , namespaceURI ] )
```

attribute-name

A CHARACTER or LONGCHAR expression evaluating to the fully qualified or unqualified name of the attribute.

attribute-value

A CHARACTER or LONGCHAR expression evaluating to the value of the attribute.

namespaceURI

A CHARACTER or LONGCHAR expression evaluating to:

- The URI of the attribute
- A zero-length string
- The Unknown value (?) if the attribute doesn't contain a namespace

SAX-writer object handle usage

Call this method to add a simple, single attribute to a start tag. You can only call this method immediately after a call to the START-ELEMENT, EMPTY-ELEMENT, INSERT-ATTRIBUTE, or DECLARE-NAMESPACE method. That is, you can only call this method when the WRITE-STATUS is SAX-WRITE-TAG. After calling this method, the status remains SAX-WRITE-TAG.

The STRICT attribute setting determines if the XML output for a SAX-writer object is well formed. However, regardless of the value of the STRICT attribute, this method fails if you do not call it following one of the methods listed in the previous paragraph.

SAX-attributes object handle usage

Call this method to add an attribute to a SAX-attributes object. If there are already attributes in the SAX-attributes object, then the new attribute is added to the end of the list. For example, if the SAX-attributes object has three attributes, then the NUM-ITEMS attribute value will be 3. After calling this method, NUM-ITEMS will be 4 and the new attribute will be located at index position 4.

You can use this method to populate a new SAX-attributes object from scratch. For example, if all the XML elements in a particular document have a default set of attributes, you can create one SAX-attributes object to hold the default set and reuse that SAX-attributes object when writing each XML element.

No validation is done on inserted data. Care must be taken to not enter duplicate attribute names. Non-unique attribute names prevent XML from being validated as well formed XML. Such a result will cause an error with the SAX-writer object during writing operations if the STRICT attribute is set to TRUE.

Caution: Use the UPDATE-ATTRIBUTE( ) to change attribute values. Using the INSERT-ATTRIBUTE( ) method with an attribute name that already exists in a SAX-attributes object results in duplicate attribute names. The resulting XML is not well-formed and may fail validation within ABL applications and within third-party XML-enabled applications.

Namespace declarations with attributes

Namespace declarations appear as attributes in an XML document. You can declare a namespace by using the INSERT-ATTRIBUTE( ) method instead of using the SAX-writer object's DECLARE-NAMESPACE( ) method. To do this, you must use the special prefix for namespaces, **xmlns**, with the name of the attribute, and the *namespaceURI* must be the value of the attribute.

```
swh:START-ELEMENT("prefix:name").
swh:INSERT-ATTRIBUTE("xmlns:prefix", "target.url").
swh:END-ELEMENT("prefix:name").
```

The previous example is equivalent to the following example, which uses the traditional method:

```
swh:START-ELEMENT("prefix:name").
swh:DECLARE-NAMESPACE("target.url", "prefix").
swh:END-ELEMENT("prefix:name").
```

Namespaces declared using the INSERT-ATTRIBUTE( ) method in this way are considered valid namespaces and will be included in namespace checking during the XML write operation. If the `xmlns` prefix does not appear, then the attribute is inserted as a regular attribute and it is not handled with the namespace checking during the XML write operation.

Similarly, namespaces that appear as such in a SAX-attributes object will also generate the appropriate namespace.

If you use *namespaceURI*, then the method resolves the prefix in the following order:

- It attempts to extract the namespace from *attribute-name*.
- It attempts to extract the namespace from a previously declared namespace.

If the method call only contains *name* and that value contains a prefix, then the SAX-writer attempts to resolve the prefix to a namespace. If it fails to resolve the namespace and the STRICT attribute is TRUE, then the method fails.

Note: The INSERT-ATTRIBUTE( ) method allows some non-printing characters to be written to XML documents. These characters (for example, #x1, #x1F) are not part of the XML Version 1.0 standard, but are allowed in order to support ABL applications.

The following is a SAX-attributes object example:

```
DEFINE VARIABLE hSAX-attributes AS HANDLE NO-UNDO.

CREATE SAX-ATTRIBUTES hSAX-attributes.

hSAX-attributes:INSERT-ATTRIBUTE( "language", "EN" ). /* index = 1 */
hSAX-attributes:INSERT-ATTRIBUTE( "year", "2006" ).   /* index = 2 */
```

See also

DECLARE-NAMESPACE( ) method, REMOVE-ATTRIBUTE( ) method, UPDATE-ATTRIBUTE( ) method

INSERT-BACKTAB( ) method
(Character interfaces only)

Moves the cursor backward to the previous four-space tab stop without affecting the text in the widget.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
INSERT-BACKTAB ( )
```

If the operation is successful, the method returns TRUE.

INSERT-BEFORE( ) method

Insert a node as a child of this document before another node (or last if the other node is unknown). This is one way to place the node into the document structure after the node has been created with the CREATE-NODE( ) or CREATE-NODE-NAMESPACE( ) method, cloned with the CLONE-NODE( ) method, or removed with the REMOVE-NODE( ) method. (Similar to the APPEND-CHILD( ) method.)

Return type: LOGICAL

Applies to: X-document object handle, X-noderef object handle

Syntax

```
INSERT-BEFORE ( x-ref-handle1 , x-ref-handle2 )
```

x-ref-handle1

The handle that represents the node to insert as a child of this document.

x-ref-handle2

A handle that represents the XML node that the node is to be inserted before. If unknown, the node will be appended as the last child.

The following code fragment demonstrates the use of the INSERT-BEFORE( ) method. hNoderefParent ends up with hNoderef and hNoderef2 in that order:

```
. . .
hDoc:CREATE-NODE(hNoderef,bufField:NAME,"ELEMENT").
hNoderefParent:INSERT-BEFORE(hNoderef,hNoderef2).
. . .
```

INSERT-FILE( ) method

Inserts the text of *filename* into the editor widget at the current location of the text cursor.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
INSERT-FILE ( filename )
```

filename

A character-string expression equal to the full or relative pathname of a file. If you specify a relative pathname, the INSERT-FILE( ) method searches PROPATH to find the file.

If the text insertion is successful, the method returns TRUE.

Note: This method replaces each horizontal tab character with eight spaces as it inserts the text into the widget.

INSERT-ROW( ) method

Inserts a blank line in an updateable browse before or after the last selected row. The blank line is a placeholder for a new record to be added through the browse. This method cannot be used with a read-only browse.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
INSERT-ROW ( BEFORE | AFTER )
```

BEFORE

Adds a new row before the current browse row. This is the default.

AFTER

Adds a new row after the current browse row.

INSERT-STRING( ) method

Inserts a string into the editor widget at the current location of the text cursor.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
INSERT-STRING ( string )
```

string

A character string expression.

If the operation is successful, the method returns TRUE.

INSERT-TAB( ) method
(Character interfaces only)

This method works differently depending on the insert mode status. If insert mode is on, it inserts one to four spaces from the current cursor position to the next four-space tab stop. If insert mode is off, it moves the cursor to the next four-space tab stop without inserting any characters.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
INSERT-TAB ( )
```

If the operation is successful, the method returns TRUE.

INSTANTIATING-PROCEDURE attribute

Returns the handle to the procedure in which an object was instantiated.

Data type: HANDLE

Access: Read-only

Applies to: ACTIVE-WINDOW system handle, Asynchronous request object handle, AUDIT-CONTROL system handle, AUDIT-POLICY system handle, BROWSE widget (browse and column), Buffer object handle, Buffer-field object handle, BUTTON widget, Call object handle, Client-principal object handle, CLIPBOARD system handle, CODEBASE-LOCATOR system handle, COLOR-TABLE system handle, COMBO-BOX widget, COMPILER system handle, CONTROL-FRAME widget, CURRENT-WINDOW system handle, Data-relation object handle, Data-source object handle, DEBUGGER system handle, DEFAULT-WINDOW system handle, DIALOG-BOX widget, DSLOG-MANAGER system handle, EDITOR widget, ERROR-STATUS system handle, FIELD-GROUP widget, FILE-INFO system handle, FILL-IN widget, FONT-TABLE system handle, FRAME widget, IMAGE widget, LAST-EVENT system handle, LITERAL widget, LOG-MANAGER system handle, MENU widget, MENU-ITEM widget, Procedure object handle, ProDataSet object handle, PROFILER system handle, Query object handle, RADIO-SET widget, RCODE-INFO handle, RECTANGLE widget, SAX-attributes object handle, SAX-reader object handle, SAX-writer object handle, SECURITY-POLICY system handle, SELECTION-LIST widget, SELF system handle, Server object handle, Server socket object handle, SESSION system handle, SLIDER widget, SOAP-fault object handle, SOAP-fault-detail object handle, SOAP-header object handle, SOAP-header-entryref object handle, Socket object handle, SOURCE-PROCEDURE system handle, Stream object handle, SUB-MENU widget, TARGET-PROCEDURE system handle, Temp-table object handle, TEXT widget, THIS-PROCEDURE system handle, TOGGLE-BOX widget, Transaction object handle, WEB-CONTEXT system handle, WINDOW widget, X-document object handle, X-noderef object handle

When applied to a widget or an object handle, this attribute returns a handle. This handle is either valid or invalid, based on the state of the instantiating procedure. The instantiating procedure for a dynamically created object might no longer be running. In this case, the handle is invalid.

Note: Because you can not be sure of the state of an instantiating procedure, never use this attribute as a chained attribute.

In general, the INSTANTIATING-PROCEDURE attribute returns the procedure handle for the procedure in which the DEFINE or CREATE statement is executed. The following are the exceptions:

- For a procedure handle or an asynchronous request object handle, INSTANTIATING-PROCEDURE returns the procedure handle for the procedure from which the RUN statement was executed.
- For a socket handle created on a socket server after a CONNECT statement, INSTANTIATING-PROCEDURE returns the procedure handle for the procedure in which the socket server connect procedure is defined.
- For a temp-table handle that is passed from a client to an application server, INSTANTIATING-PROCEDURE returns the procedure handle for the first procedure run on the application server.
- For a temp-table handle that is received from an application server by a client, INSTANTIATING-PROCEDURE returns the procedure handle for the procedure that called the application server.
- For implicitly created handles, INSTANTIATING-PROCEDURE returns the procedure handle where the implicitly created object is returned (such as the SAX-attributes object in the StartElement callback).
- For the COM-SELF system handle, which is not a widget handle, INSTANTIATING-PROCEDURE returns a run-time error.

Although this attribute applies to all widgets and handles, not all system handles are associated with an instantiating procedure and do not return a procedure handle. For these system handles, this attribute returns the unknown value (?). The following system handles always return the unknown value (?):

AUDIT-CONTROL system handle	CLIPBOARD system handle
CODEBASE-LOCATOR system handle	COLOR-TABLE system handle
COMPILER system handle	DEBUGGER system handle

DSLOG-MANAGER system handle	ERROR-STATUS system handle
FILE-INFO system handle	FOCUS system handle
FONT-TABLE system handle	LAST-EVENT system handle
LOG-MANAGER system handle	RCODE-INFO system handle
SECURITY-POLICY system handle	SESSION system handle
WEB-CONTEXT system handle	–

INTERNAL-ENTRIES attribute

A comma-separated list containing the names of all internal procedures and user-defined functions defined in the procedure associated with the specified handle. Returns the Unknown value (?) for a Web service procedure or proxy procedure.

Data type: CHARACTER

Access: Read-only

Applies to: Procedure object handle, SOURCE-PROCEDURE system handle, TARGET-PROCEDURE system handle, THIS-PROCEDURE system handle

If you supply a handle to a procedure that defines no internal procedures or user-defined functions, the value of INTERNAL-ENTRIES is the null string ("").

Note: The list provided by INTERNAL-ENTRIES does not contain the name of any internal procedure defined using the PROCEDURE statement's PRIVATE option. Similarly, the list does not contain the name of any user-defined function defined using the FUNCTION statement's PRIVATE option. However, if the procedure object handle is equal to THIS-PROCEDURE, INTERNAL-ENTRIES returns all internal procedures and user-defined functions, including those defined as PRIVATE.

INVOKE( ) method (Handle)

Lets you do the following dynamically:

- Invoke an external procedure, internal procedure, or user-defined function
- Invoke a Windows DLL routine or Unix shared library routine
- Get or set an object handle attribute
- Run an object handle method

Return type: None (Similar to the RUN statement.)

Applies to: Call object handle

Syntax

```
INVOKE( )
```

To determine what action to take, INVOKE( ) examines the CALL-NAME and CALL-TYPE attributes.

Before you execute INVOKE( ), you must set the CALL-NAME attribute.

The following table describes what INVOKE( ) does for each call type.

Table 83: What INVOKE( ) does for each call type

For this call type . . .	INVOKE( ) . . .
PROCEDURE-CALL-TYPE (the default)	Follows the rules of the RUN statement to determine what to invoke and whether to invoke external or internal procedures.
FUNCTION-CALL-TYPE	Follows the rules of the DYNAMIC-FUNCTION function.
GET-ATTR-CALL-TYPE	Follows the rules of *widget*:*attribute*. **Note:** The IN-HANDLE attribute must be set before INVOKE( ) is executed.
SET-ATTR-CALL-TYPE	Sets the attribute specified by the CALL-NAME attribute to the first parameter specified by the SET-PARAMETER( ) method. **Note:** The IN-HANDLE attribute must be set before INVOKE( ) is executed.

If the PERSISTENT attribute is TRUE or PROCEDURE-TYPE attribute is set to "PERSISTENT" (which is valid only for invoking an external procedure), the procedure runs persistently, and when INVOKE( ) returns, IN-HANDLE contains a handle to the persistent procedure. If PROCEDURE-TYPE is set to "SINGLE-RUN" or "SINGLETON", IN-HANDLE contains a handle to the single-run or singleton procedure, respectively. Note that unlike persistent procedures, procedures run as single-run or singleton cannot have parameters in their main blocks.

Before you dynamically invoke an external procedure that is remote—that is, one that resides on an application server—you must set the SERVER attribute to the handle of the application server.

When INVOKE( ) starts executing, it examines the NUM-PARAMETERS attribute. If NUM-PARAMETERS is nonzero, INVOKE( ) uses each parameter set by the SET-PARAMETER( ) method, even one set during a previous use of the call object.

Note: To clear all parameters, even those set during a previous use of the call object, set the NUM-PARAMETERS attribute to zero.

No parameters are evaluated during INVOKE( ) processing. Parameters are evaluated only during SET-PARAMETER( ) processing.

When the invoked routine starts, if any parameter indicated by the NUM-PARAMETERS attribute has not been set, you will get an error message.

For more information on INVOKE( ), see the reference entries for the RUN statement and the DYNAMIC-FUNCTION function.

Note: INVOKE( ) cannot occur within an expression.

IS-CLASS attribute

Indicates whether an r-code file defines a class or a procedure.

Data type: LOGICAL

Access: Read-only

Applies to: RCODE-INFO handle

The IS-CLASS attribute is TRUE if the currently executing r-code file defines a class that has been instantiated using the NEW function (classes). If the attribute is FALSE, the r-code file defines a procedure.

IS-JSON attribute

Returns whether a JSON string was posted to the transaction server. The AVM determines the value by checking if the content-type HTTP header is either "application/json" or "text/json".

Data type: LOGICAL

Access: Read-only

Applies to: WEB-CONTEXT system handle

IS-MULTI-TENANT attribute

Indicates whether the table of the buffer is multi-tenant.

Data type: LOGICAL

Access: Read-only

Applies to: Buffer object handle

If the buffer is for a multi-tenant table, IS-MULTI-TENANT is TRUE. Otherwise, the attribute returns FALSE.

See also

IS-DB-MULTI-TENANT function

IS-OPEN attribute

Indicates whether a transaction or query object is open.

Data type: LOGICAL

Access: Read-only

Applies to: Query object handle, Transaction object handle

The IS-OPEN attribute is TRUE if the specified database transaction or query object is active.

For transaction handles, this attribute is identical to the TRANSACTION function.

IS-PARAMETER-SET attribute

Indicates whether you have already set a particular parameter.

Data type: LOGICAL

Access: Read-only

Applies to: Call object handle

Syntax

```
IS-PARAMETER-SET ( parameter-number )
```

parameter-number

An integer expression indicating the order of the parameter, where 1 represents the first parameter, 2 represents the second parameter, and so on.

If the parameter is set, IS-PARAMETER-SET returns TRUE. Otherwise, it returns FALSE. The default is FALSE.

IS-PARTITIONED attribute

Indicates whether the table of the buffer is partitioned.

Data type: LOGICAL

Access: Read-only

Applies to: Buffer object handle

If the buffer is for a partitioned table, IS-PARTITIONED returns TRUE. Otherwise, the attribute returns FALSE.

IS-ROW-SELECTED( ) method

Returns TRUE if a specified row in the browse viewport is currently selected.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
IS-ROW-SELECTED ( n )
```

n

An integer expression that specifies a selected row within the browse viewport.

The AVM maintains a numbered list of selected rows, starting at 1.

IS-SELECTED( ) method

Returns TRUE if a specified item in a selection list is currently selected. Otherwise, the method returns FALSE.

Return type: LOGICAL

Applies to: SELECTION-LIST widget

Syntax

```
IS-SELECTED ( list-item | list-index )
```

list-item

A character-string expression that specifies a single value in the selection list.

list-index

An integer expression that specifies the ordinal position (first, second, third, etc.) of an entry in the selection-list.

IS-XML attribute

Returns whether an XML document was posted to the transaction server.

Data type: LOGICAL

Access: Read-only

Applies to: WEB-CONTEXT system handle

ITEMS-PER-ROW attribute

How to format multiple items written to the system clipboard using the CLIPBOARD handle.

Data type: INTEGER

Access: Readable/Writeable

Applies to: CLIPBOARD system handle

This attribute has meaning only when the CLIPBOARD:MULTIPLE attribute is TRUE, and has no effect when reading data from the clipboard. For more information, see the reference entry for the CLIPBOARD system handle.

KEEP-CONNECTION-OPEN attribute

Indicates whether WebClient should keep any server connection, that it creates, open after downloading a file (TRUE) or not (FALSE).

Data type: LOGICAL

Access: Read-only

Applies to: CODEBASE-LOCATOR system handle

KEEP-FRAME-Z-ORDER attribute

The overlay order of the frames in a window.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WINDOW widget

If the KEEP-FRAME-Z-ORDER attribute is set to TRUE, the AVM ignores the default frame overlay behavior.

By default, the AVM moves the frame that contains the field with focus to the top. When you set the KEEP-FRAME-Z-ORDER attribute to TRUE, you are responsible for maintaining the overlay order of the frames using the MOVE-TO-TOP( ) and MOVE-TO-BOTTOM( ) methods. You should always set this attribute to TRUE when you use the MOVE-TO-TOP( ) and MOVE-TO-BOTTOM( ) methods.

The default value is FALSE.

KEEP-SECURITY-CACHE attribute

Indicates whether WebClient saves the values of the attributes in the security cache between sessions (TRUE) or not (FALSE), as requested by the user. The default value is FALSE.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: CODEBASE-LOCATOR system handle

The following attributes represent the security cache:

- APPSERVER-INFO
- APPSERVER-PASSWORD
- APPSERVER-USERID
- URL-PASSWORD
- URL-USERID

- KEEP-SECURITY-CACHE

These attributes are readable and writeable.

KEY attribute

Indicates whether the field corresponding to a buffer-field participates in an index.

Data type: LOGICAL

Access: Read-only

Applies to: Buffer-field object handle

KEYS attribute

Returns a comma-separated list of key fields for a buffer.

Data type: CHARACTER

Access: Read-only

Applies to: Buffer object handle, Data-source object handle

For a data-source object buffer, this attribute returns a comma-separated list of key fields defined in an associated KEYS clause for the specified buffer. If there are no defined key fields, this attribute returns a comma-separated list of key fields in the buffer's unique primary index (if any). If there are no defined key fields and no unique primary index, this attribute returns the string "ROWID".

Syntax

Following is the syntax for accessing this attribute for a data-source object buffer:

```
data-source-handle:KEYS ( buffer-sequence-number )
```

data-source-handle

The handle to the data-source object.

buffer-sequence-number

An integer that represents the sequence number of a buffer in the list of buffers for the data-source object. Specify *buffer-sequence-number* to identify a buffer in the data-source object when the data-source object is defined against more than one database table buffer.

Note: Sequence numbers for buffers of a query start at one, where one represents the top level and subsequent numbers represent lower levels of a join, if any.

Syntax

Following is the syntax for accessing this attribute for a buffer directly:

```
buffer-handle:KEYS
```

buffer-handle

The handle to the buffer.

LABEL attribute

The label of a widget or the name of a low-level event.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget (column), Buffer-field object handle, BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, MENU-ITEM widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, LAST-EVENT system handle

For the LAST-EVENT handle, this attribute is readable only.

For a widget, the LABEL attribute specifies the label for the widget. For some data representation widgets, it is actually the SCREEN-VALUE attribute value of the literal or text widget that is assigned to the SIDE-LABEL-HANDLE attribute of the specified data representation widget. For more information, see the SIDE-LABEL-HANDLE attribute reference entry.

For the LAST-EVENT handle, the LABEL attribute returns the names of low-level events based on the EVENT-TYPE attribute value. For EVENT-TYPE = "KEYPRESS", this attribute returns key label events, such as "F1" or "ESC". It also returns the key labels of any keys that trigger key function events (returned by the FUNCTION attribute).

For EVENT-TYPE = "MOUSE", this attribute returns low-level events for both portable and three-button mouse event types, such as "SELECT-MOUSE-UP" (portable) or "LEFT-MOUSE-UP" (three-button). It also returns the names of the low-level mouse actions that trigger any high-level mouse events (returned by the FUNCTION attribute).

For EVENT-TYPE = "PROGRESS", this attribute returns the same high-level event name returned by the FUNCTION attribute unless the ABL event is triggered by a key press. In this case, it returns the key label of the key that triggered the event.

Note: When the AUTO-RESIZE attribute is set to TRUE, the AVM resizes button and toggle-box widgets with run-time changes to the LABEL attribute.

LABEL-BGCOLOR attribute

The color number of the background color for a column label or all column labels in a browse widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column)

LABEL-DCOLOR attribute
(Character interfaces only)

The color number of the display color for a column label or all column labels in a browse widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column)

LABEL-FGCOLOR attribute

The color number of the foreground color for a column label or all column labels in a browse widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column)

LABEL-FONT attribute

The font for a column label or all column labels in a browse widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column)

LABELS attribute

Indicates whether a label appears with the widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, TEXT widget

This attribute applies to static frames.

This attribute is writeable for the browse widget only. If the LABELS attribute is set to FALSE for a browse widget, no column headers will appear with the browse.

If the LABELS attribute is FALSE for a combo-box, editor, fill-in, or text widget, no label appears with the widget. If the attribute is FALSE for a frame, the NO-LABELS option is specified in the frame phrase for the frame and no labels are displayed with field-level widgets within the frame.

LABELS-HAVE-COLONS attribute

Indicates whether the r-code was compiled with the `-nocolon` startup parameter in effect. (See the *Startup Command and Parameter Reference* for more information about `-nocolon`.)

Data type: LOGICAL

Access: Read-only

Applies to: RCODE-INFO handle

If the r-code file was compiled with the `-nocolon` startup parameter in effect, colons are not appended to side labels for static widgets, and this attribute returns FALSE. If the r-code file was not compiled with the `-nocolon` startup parameter in effect, this attribute returns TRUE.

LANGUAGES attribute

A comma-separated list of all languages compiled into the r-code file specified by the RCODE-INFO:FILE-NAME attribute.

Data type: CHARACTER

Access: Read-only

Applies to: RCODE-INFO handle

If the r-code file was compiled with only the default language, LANGUAGES returns an empty string.

LARGE attribute
(Windows only; Graphical interfaces only)

Indicates whether a Windows editor widget can hold 32K of text. Non-Windows platforms ignore this attribute.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: EDITOR widget

When LARGE is FALSE, the Windows editor widget can hold 32K of text. When LARGE is TRUE, the Windows editor widget can hold over 64K of text—the precise limit depends on available resources.

You can set this attribute only before the editor widget is realized.

Note: In character interfaces, the editor widget can hold large amounts of text by default. Therefore, character interfaces do not need separate "large" and "small" editors.

LARGE-TO-SMALL attribute

The default numeric range that a slider can display is small (minimum) to large (maximum). The LARGE-TO-SMALL option allows you to override this default behavior as follows:

- When the slider is positioned horizontally, the left-most position on the trackbar displays the maximum value and the right-most position displays the minimum value.
- When the slider is positioned vertically, the bottom-most position on the trackbar displays the maximum value and the top-most position displays the minimum value.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SLIDER widget

LAST-ASYNC-REQUEST attribute

Returns the last entry in the list of all current asynchronous request handles for the specified application server or Web service that have been created in the current session.

Data type: HANDLE

Access: Read-only

Applies to: Server object handle

If there are no asynchronous request handles for the specified server, LAST-ASYNC-REQUEST returns the Unknown value (?).

LAST-BATCH attribute

Indicates whether a FILL operation on a ProDataSet temp-table retrieved the last batch of rows in its associated query.

Note: You typically use the LAST-BATCH attribute in conjunction with the BATCH-SIZE attribute. It is best to use the LAST-BATCH attribute with a top-level ProDataSet temp-table. If you use the LAST-BATCH attribute with the BATCH-SIZE attribute for a child temp-table, the parent (or ancestor) must have only one row, because the AVM restarts the BATCH-SIZE counter on a child temp-table for each parent record (as opposed to once per child temp-table).

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Buffer object handle

See also

BATCH-SIZE attribute, FILL( ) method

LAST-CHILD attribute

The handle of the last widget created in the container widget or the current session.

Data type: HANDLE

Access: Read-only

Applies to: DIALOG-BOX widget, FIELD-GROUP widget, FRAME widget, MENU widget, SESSION system handle, SUB-MENU widget, WINDOW widget

You can use the LAST-CHILD option to find the last entry in a list of all frames and dialog boxes in a window, all field groups in a frame or dialog box, all widgets in a field group, all menu items in a menu or submenu, or all windows in an ABL session (SESSION system handle). After finding the last entry, you can find the remaining entries in the list by using each widget's PREV-SIBLING attribute.

LAST-FORM attribute
(Windows only; GUI for .NET only)

The object reference for the last .NET `Progress.Windows.IForm` or ABL window `FormProxy` (`Progress.Windows.FormProxy`) object in the list of all valid OpenEdge form objects created in the current ABL session.

Data type: Progress.Windows.IForm interface

Access: Read-only

Applies to: SESSION system handle

Once you get the last object reference in the list, you can use the PrevForm property in the `Progress.Windows.IForm` interface to get the previous entry in the list of OpenEdge form object references.

Note: To obtain the last class-based object in the list of all valid ABL and .NET objects created in the current ABL session, use the LAST-OBJECT attribute.

If you instantiate a `Progress.Windows.IForm` in a non-GUI ABL session, the object does not appear in the list of OpenEdge form objects, but only in the list of valid class-based objects.

See also

FIRST-FORM attribute, FIRST-OBJECT attribute, LAST-OBJECT attribute, PrevForm property, Progress.Windows.FormProxy class, Progress.Windows.IForm interface

LAST-OBJECT attribute

The object reference for the last class instance in the list of all valid ABL and .NET class instances created in the current ABL session. If there are no class instances in the current session, this attribute returns the Unknown value (?).

Data type: Progress.Lang.Object

Access: Read-only

Applies to: SESSION system handle

Once you get the last object reference in the list, you can use the PREV-SIBLING property in the Progress.Lang.Object class to get the previous entry in the list of ABL and .NET object references.

There are two points worth noting about this attribute with garbage collection enabled:

- The object references maintained by the LAST-OBJECT attribute and the PREV-SIBLING property do not count as references for garbage collection. That is, if a class instance is referenced only on the session object chain, it is available for automatic garbage collection.

- When an ABL object that inherits from a .NET object is on the session object chain, it can have circular references that prevent it from being deleted (as with pure ABL objects). When there are no more ABL references to such an ABL-derived .NET object, the object is removed from the object chain, but may still be instantiated because of references to it from .NET. If the application re-obtains a reference to the object in ABL by getting it from .NET, the reference is again placed on the object chain.

Note: To obtain the last .NET form or ABL FormProxy object in the list of all valid form objects created in the current ABL session, use the LAST-FORM attribute.

See also

FIRST-OBJECT attribute, LAST-FORM attribute, PREV-SIBLING property, Progress.Lang.Object class

LAST-OF( ) method

Returns TRUE if the current iteration of the query predicate FOR EACH . . . BREAK BY. . . is the last iteration for a new break group.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
LAST-OF ( level )
```

level

An integer expression that indicates a BREAK BY group, where 0 is the entire query, 1 is the first BREAK BY, 2 is the second BREAK BY, and so on.

The following example shows how the LAST-OF( ) method is used to identify the last record of a break group:

```
DEFINE QUERY qCustomer FOR Customer SCROLLING.

QUERY qCustomer:QUERY-PREPARE("FOR EACH Customer WHERE Customer.CustNum > 50
  BREAK BY Customer.Country BY Customer.Comments").
QUERY qCustomer:QUERY-OPEN.

REPEAT WITH TITLE "Customers Break By Country and Comments":
  GET NEXT qCustomer.
  IF NOT AVAILABLE Customer THEN LEAVE.
  DISPLAY
    Customer.Country FORMAT "x(10)"
    Customer.Comments FORMAT "x(20)"
    QUERY qCustomer:FIRST-OF(0) LABEL "First"
    QUERY qCustomer:LAST-OF(0) LABEL "Last"
    QUERY qCustomer:FIRST-OF(1) LABEL "First Country"
    QUERY qCustomer:LAST-OF(1) LABEL "Last Country"
    QUERY qCustomer:FIRST-OF(2) LABEL "First Company"
    QUERY qCustomer:LAST-OF(2) LABEL "Last Company".
END.
```

See also

FIRST-OF( ) method, OPEN QUERY statement, QUERY-PREPARE( ) method

LAST-PROCEDURE attribute

For an application server, returns a handle to the last entry in the list of remote persistent, single-run, and singleton procedures that are active on the connected application server. For Web services, returns a handle to the last entry in the list of procedure objects associated with the Web service.

Data type: HANDLE

Access: Read-only

Applies to: Server object handle, SESSION system handle

If the current session has no active persistent procedures or the application server has no active remote persistent, single-run, and singleton procedures, LAST-PROCEDURE returns the Unknown value (?). To find the previous procedure given the last, access the PREV-SIBLING attribute of the procedure handle you just got.

Note: The PERSISTENT attribute of a remote persistent procedure returned by LAST-PROCEDURE is TRUE only if the RUN statement that executes the procedure includes the PERSISTENT option. The PERSISTENT attribute is FALSE if the persistent procedure was invoked with the SINGLE-RUN or SINGLETON option of the RUN statement.

For information on creating persistent, single-run, or singleton procedures, see the RUN statement. To check a handle for validity, use the VALID-HANDLE function.

LAST-SERVER attribute

A handle to the last entry in the list of server handles for the current ABL session. This includes both application server objects and Web service server objects.

Data type: HANDLE

Access: Read-only

Applies to: SESSION system handle

Returns the handle associated with the last entry in the list of all server handles created in the current session. If the current session has no server handles, LAST-SERVER has the Unknown value (?). For more information on server handles, see the Server object handle reference entry.

LAST-SERVER-SOCKET attribute

A handle to the last entry in the list of all valid server socket handles created in the current session. If there are no server socket handles in this session, LAST-SERVER-SOCKET returns the Unknown value (?).

Data type: HANDLE

Access: Read-only

Applies to: SESSION system handle

LAST-SOCKET attribute

A handle to the last entry in the list of all valid socket handles created in the current session. If there are no socket handles in this session, LAST-SOCKET returns the Unknown value (?).

Data type: HANDLE

Access: Read-only

Applies to: SESSION system handle

LAST-TAB-ITEM attribute

The last widget in the tab order of a field group.

Data type: HANDLE

Access: Readable/Writeable

Applies to: FIELD-GROUP widget

When you set this attribute, the assigned widget is moved to the last tab position, following the widget that was previously at this position. Other widgets in the field group maintain their same relative tab positions.

To set the attribute, you must assign it the handle of a field-level widget or frame that can receive focus from a TAB event and that is also a child of the field group to which the attribute applies. If the LAST-TAB-ITEM attribute is not set (is the Unknown value (?)), the default last tab position goes to the widget identified by the LAST-CHILD attribute of the field group.

For more information on how frames owned by a field group participate in the tab order of that field group, see the FRAME widget reference entry.

Note: Any tab reordering that you do with this attribute can be reset by a subsequent ENABLE statement unless you define the frame that owns the field group with the KEEP-TAB-ORDER option. For more information, see the ENABLE statement and Frame phrase reference entries.

Left property
(Windows only; Graphical interfaces only)

The horizontal position of the control-frame and control-frame COM object from the left side of the parent container widget, in pixels.

Return type: INTEGER

Access: Readable/Writeable

Applies to: CONTROL-FRAME widget, COM object

Setting this value changes the COLUMN attribute and X attribute of the corresponding control-frame widget to an equivalent value.

Note: References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the Accessing COM object properties and methods.

LENGTH attribute

The length (number of characters) of the current content of the editor widget.

Data type: INTEGER

Access: Read-only

Applies to: EDITOR widget

If the editor widget is not already realized and you reference its LENGTH attribute, the AVM realizes the widget.

In Windows, both the regular editor and the large editor support LENGTH.

LIBRARY attribute

Specifies the name of a Windows Dynamic Link Library (DLL) or a UNIX shared library.

Data type: CHARACTER

Access: Readable/Writable

Applies to: Call object handle

This attribute is required for dynamic invocation of a DLL or shared library. The AVM verifies the validity and existence of the LIBRARY value at run time when the call object's INVOKE( ) method is called.

If LIBRARY specifies a full path name, the AVM searches only that path for the shared library. Otherwise, the AVM first determines if the shared library is loaded in memory. If it is not loaded in memory, the AVM looks for the shared library in the current working directory. If the shared library cannot be found at this point:

- On Windows the AVM uses the standard Windows search algorithm to find the DLL, the same algorithm used to find Windows executable (`.exe`) files
- On UNIX the AVM continues searching for the shared library using the ABL PROPATH

See also

INVOKE( ) method (Handle)

LIBRARY-CALLING-CONVENTION attribute

Specifies the calling convention for a Windows Dynamic Link Library (DLL) or UNIX shared library.

Data type: CHARACTER

Access: Readable/Writable

Applies to: Call object handle

The following table lists the valid, case-sensitive, keyword constant values for the LIBRARY-CALLING-CONVENTION attribute.

Table 84: Valid LIBRARY-CALLING-CONVENTION attribute values

Keyword constant	Description
STDCALL	Use the standard Windows calling convention when accessing the routine. This is the default value.
CDECL	Use the C calling convention when accessing the routine.

The AVM returns a run-time error if a value other than "CDECL" or "STDCALL" is assigned to this attribute.

See also

INVOKE( ) method (Handle), LIBRARY attribute, ORDINAL attribute

LINE attribute

The current logical line number (iteration number) of the frame.

Data type: INTEGER

Access: Read-only

Applies to: FRAME widget

This attribute applies to down frames only.

This attributes is equivalent to the FRAME-LINE function. For more information, see the FRAME-LINE function reference entry.

LIST-ITEM-PAIRS attribute

A list of the label-value pairs associated with a combo box or selection list. The list is delimiter-separated.

Note: The LIST-ITEM-PAIRS attribute applies only to combo boxes and selection lists whose entries consist of label-value pairs. For combo boxes and selection lists whose entries consist of single items, use the LIST-ITEMS attribute.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget (column), COMBO-BOX widget, SELECTION-LIST widget

The value of the delimiter depends on the value of the DELIMITER attribute, which is a comma by default.

For browses, this attribute applies only to combo-box browse columns.

LIST-ITEM-PAIRS provides a list like the following:

```
"Red,1,Blue,2,Green,3"
```

LIST-ITEMS attribute

A list of the items associated with a combo box or selection list. The list is delimiter-separated.

Note: The LIST-ITEMS attribute applies only to combo boxes and selection lists whose entries consist of single items. For combo boxes and selection lists whose entries consist of label-value pairs, use the LIST-ITEM-PAIRS attribute.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget (column), COMBO-BOX widget, SELECTION-LIST widget

The value of the delimiter depends on the value of the DELIMITER attribute, which is comma by default.

For browses, this attribute applies only to combo-box browse columns.

LIST-ITEMS provides a list like the following:

```
"Red,Blue,Green"
```

LIST-PROPERTY-NAMES( ) method

Returns a comma-separated list of all application-defined properties stored in the client-principal object. The client-principal object can be sealed or unsealed.

Return type: CHARACTER

Applies to: Client-principal object handle

Syntax

```
LIST-PROPERTY-NAMES( )
```

You can also use the GET-PROPERTY( ) method to get the value of a single property stored in the client-principal object.

The following code fragment illustrates how to use the LIST-PROPERTY-NAMES( ) method:

```
DEFINE VARIABLE hCP   AS HANDLE    NO-UNDO.
DEFINE VARIABLE vProp AS CHARACTER NO-UNDO.
. . .
CREATE CLIENT-PRINCIPAL hCp.
. . .
vProp = hCP:LIST-PROPERTY-NAMES( ).
DISPLAY vProp FORMAT "X(70)".
```

See also

CAN-DO function, ENTRY function, GET-PROPERTY( ) method, NUM-ENTRIES function, SET-PROPERTY( ) method

LISTINGS attribute

Determines whether or not the profiler automatically generates a debug listing files for the ABL code being profiled. Requires access to source code, not intended for use in production systems. The default value is FALSE.

Data type: LOGICAL

Access: Readable/Writable

Applies to: PROFILER system handle

The profiler generates a debug listing file the first time that a procedure is executed. Therefore, if PROFILER:LISTINGS attribute is FALSE the first time a procedure is executed, then setting it to TRUE at a later time does not cause the profiler to generate the listing file for that procedure. In other words, the profiler only registers a module, using a unique identifier, the first time the module executes. The profiler only generates debug listing files for modules that are external procedures. If PROFILER:LISTINGS is FALSE when a procedure first executes, then the profiler never tries to generate a debug listing file for that procedure. It fails if the source files are not accessible. The source files that make up a procedure must be in the PROPATH in order for the profiler to generate a debug listing file. If the profiler cannot generate a debug listing file for a given procedure, then the profiler treats the procedure as though the PROFILER:LISTINGS setting was FALSE when that procedure first executes. The only way to get a Progress session to try again to generate the debug listing for a procedure is to disable and then re-enable the profiler by setting PROFILER:ENABLED to FALSE (which causes Progress to discard its profiler registry of procedures) and then set it to TRUE again.

LITERAL-QUESTION attribute

Lets you specify how the AVM interprets a quoted character value during assignment into the BUFFER-VALUE attribute for a character BUFFER-FIELD object. That is, whether the AVM treats the quoted character value as a literal or non-literal character value.

Return type: LOGICAL

Access: Readable/Writeable

Applies to: Buffer-field object handle

When TRUE, the AVM treats a quoted character value as a literal character value. That is, it does not remove enclosing quotes, trailing blanks, or formatting insertion characters.

When FALSE, the default value, the AVM treats a quoted character value as a non-literal character value. That is, it removes enclosing quotes, trailing blanks, and formatting insertion characters. For example:

- The AVM treats "`abc  `" as `abc`.
- The AVM treats a quoted question mark character ("?") as the Unknown value (?).

You can use the Literal Question (`-literalquestion`) startup parameter to change the default value of the LITERAL-QUESTION attribute to TRUE (which would otherwise be FALSE). For more information about this startup parameter, see *Startup Command and Parameter Reference*.

See also

BUFFER-VALUE( ) method

LOAD( ) method (Handle)

Loads an XML document into memory, parses it, and makes its contents available in ABL as an X-document object.

This method sets the ENCODING attribute for the XML document to the encoding name specified in the XML document's encoding declaration.

Return type: LOGICAL

Applies to: X-document object handle

Syntax

```
LOAD ( mode , { file | memptr | longchar } , validate )
```

mode

A character expression that evaluates to one of the following: "FILE", "MEMPTR", or "LONGCHAR".

file

A character expression that represents the name of a file that contains the source XML text. You can specify a relative pathname, an absolute pathname, or an HTTP URL. Any relative pathname is relative to the current working directory. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

memptr

A MEMPTR variable that contains the source XML text. The size of the MEMPTR variable should match the size of the XML text.

longchar

A LONGCHAR variable that contains the source XML text. The size of the LONGCHAR variable should match the size of the XML text.

validate

A logical expression where TRUE indicates that the parser should validate the document's logical structure with respect to its Document Type Definition (DTD). Note that even if validation against the DTD is not specified, the document's physical structure is still validated. If this expression is TRUE, then the parser will also validate against any XML Schema file references in the XML document or specified in the SCHEMA-LOCATION and NONAMESPACE-SCHEMA-LOCATION attribute.

The following code fragment creates a parse tree of XML nodes and validates its structure:

```
DEFINE VARIABLE hDoc AS HANDLE NO-UNDO.

CREATE X-DOCUMENT hDoc.

hDoc:LOAD("file", "memo.xml", TRUE).
. . .
```

See also

ADD-SCHEMA-LOCATION( ) method, NONAMESPACE-SCHEMA-LOCATION attribute, SCHEMA-LOCATION attribute, SCHEMA-PATH attribute

---

LoadControls( ) method
(Windows only; Graphical interfaces only)

Loads the control from a specified control file into the specified control-frame. This method loads the specified control along with all of its design-time property values.

Return type: None

Applies to: CONTROL-FRAME widget, COM object

Syntax

```
LoadControls ( control-filename , control-frame-name )
```

control-filename

The name and extension of a control (`.wrx`) file associated with the current external procedure that is created by the AppBuilder at design time. The filename can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

control-frame-name

A character-string expression that specifies the section of the control file that contains the control. Typically, this section name is also the name of the control-frame defined by the AppBuilder at design time.

References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the Accessing COM object properties and methods.

This method loads multiple controls into a control-frame.

LOAD-DOMAINS( ) method

Loads registered domains from the specified (and connected) OpenEdge RDBMS into the ABL session domain registry. The AVM uses this registry to validate client-principal objects and set the user identity through a user authentication or single sign-on (SSO) operation for the session and its available database connections using the SET-CLIENT( ) method on the SECURITY-POLICY system handle, and for database connections using the SET-DB-CLIENT function—but only for connected databases that are configured to use the session (application) registry.

Return type: LOGICAL

Applies to: SECURITY-POLICY system handle

Syntax

```
LOAD-DOMAINS ( integer-expression | logical-name | alias )
```

integer-expression

The sequence number of a connected database from which to load registered domains. For example, invoking `LOAD-DOMAINS(2)` loads registered domains from the second database.

logical-name or *alias*

The logical name or alias of a connected database from which to load registered domains. These forms require a quoted character string or a character expression.

ABL raises ERROR if you:

- Pass the parameter as the Unknown value (?)
- Pass an *integer-expression*, *logical-name*, or *alias* that evaluates to the Unknown value (?)
- Specify an *integer-expression*, *logical-name*, or *alias* that does not correspond to a connected database
- Call LOAD-DOMAINS( ) a second or additional time and the *integer-expression*, *logical-name*, or *alias* does not correspond to the database used in the initial LOAD-DOMAINS( ) call

If successful, this method returns TRUE. Otherwise, it returns FALSE and you can find the cause of the failure using the ERROR-STATUS system handle.

You cannot call this method if the LOCK-REGISTRATION( ) method is successfully called. Otherwise, you can call this method multiple times per session, as long as the database used in the initial call maintains an unbroken connection and you use the same database for all subsequent calls. (OpenEdge identifies the database using a combination of the full directory path to the `.db` file and the database creation date.) Thus, each subsequent call to this method completely replaces the previous contents of the session registry.

Without a session registry loaded using this method (or the REGISTER-DOMAIN( ) method), any attempt to set an identity using the session registry fails and returns a message.

For more information on domains and domain registries, see *Learn about Identity Management*.

See also

REGISTER-DOMAIN( ) method

LOAD-ICON( ) method
(Windows only; Graphical interfaces only)

Loads the icon you want from a specified file for display in the title bar of an ABL window, in the task bar, and when selecting a program using **ALT+TAB**. This method can accommodate icons formatted as small size (16x16) icons, regular size (32x32) icons, or both.

An icon file might contain multiple icons. In those instances when multiple icons are in a file, this method uses the 32x32 icon, if one exists, from the file that you specified. However, if a 32x32 icon does not exist, it uses the first icon in the file.

If the load is successful, this method returns TRUE.

Return type: LOGICAL

Applies to: WINDOW widget

Syntax

```
LOAD-ICON ( icon-filename [ , n] )
```

icon-filename

A character-string expression that specifies a full or relative pathname for a file that contains the icon that you want to load for display in the title bar of a window and in the task bar. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

In Windows, you can specify a URL pathname for *icon-filename*. If you specify a fully-qualified URL, LOAD-ICON( ) loads the icon file directly without searching directories or URLs in PROPATH. Valid URL protocols include HTTP and HTTPS.

Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, the LOAD-ICON( ) method continues searching with the next PROPATH entry.

If you specify URL pathnames on the PROPATH and your application repeatedly uses the LOAD-ICON( ) method with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the icon files. Use this value to create a fully-qualified URL pathname for *icon-filename* and avoid repeated searches of the PROPATH.

If you specify an empty string ("") for *icon-filename*, this method removes the most recently loaded icon.

n

An integer expression that specifies the position of the icon within the file. Only use this expression if you want to override the default behavior.

For example, the following invocation of LOAD-ICON( ) finds the second icon in the icon file `file.ico` and loads it:

```
LOAD-ICON("file.ico", 2).
```

The following invocation removes the previously loaded icon:

```
LOAD-ICON("").
```

You typically use this method to load a large icon, primarily for use in the **ALT+TAB** window for switching applications. ABL has no means of specifying a default icon for this purpose, and Windows provides a generic icon if you do not specify one.

.NET forms allow you to specify a default icon that works for this purpose using the form Icon property, which in OpenEdge is provided by the Progress.Windows.Form class. You can then change this property as required for different uses.

You can maintain uniform icons across .NET forms and ABL windows in an application by using this method to load the same large icon you are using to set the Icon property for .NET forms.

See also

LOAD-SMALL-ICON( ) method

LOAD-IMAGE( ) method
(Windows only; Graphical interfaces only)

Reads the image contained in a specified file. When applied to a button widget, the image is used for the button in its up state, and also for its down state if a separate down state image is not specified. For buttons, this is equivalent to the LOAD-IMAGE-UP( ) method.

Return type: LOGICAL

Applies to: BUTTON widget, IMAGE widget

Syntax

```
LOAD-IMAGE ( filename [ , x-offset , y-offset , width , height ] )
```

filename

A character-string expression that specifies a full or relative pathname for a file that contains an image. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

x-offset

An integer expression that specifies the pixel along the x-axis at which to begin reading from the image file.

y-offset

An integer expression that specifies the pixel along the y-axis at which to begin reading from the image file.

width

An integer expression that specifies the number of pixels along the x-axis to read from the image file.

height

An integer expression that specifies the number of pixels along the y-axis to read from the image file.

The image is not displayed until the widget is realized. If the read is successful, the method returns TRUE.

In Windows, you can specify a URL pathname. If you specify a fully-qualified URL, LOAD-IMAGE( ) loads the image file directly without searching directories or URLs in PROPATH. Valid URL protocols include HTTP and HTTPS.

Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, the LOAD-IMAGE( ) method continues searching with the next PROPATH entry.

If you specify URL pathnames on the PROPATH and your application repeatedly uses the LOAD-IMAGE( ) method with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the image files. Use this value to create a fully-qualified URL pathname for *filename* and avoid repeated searches of the PROPATH.

LOAD-IMAGE-DOWN( ) method
(Windows only; Graphical interfaces only)

Reads the image contained in a specified file. The image is used for the button in its down state only.

Return type: LOGICAL

Applies to: BUTTON widget

Syntax

```
LOAD-IMAGE-DOWN ( filename [ , x-offset , y-offset , width , height ] )
```

filename

A character-string expression that specifies a full or relative pathname for a file that contains an image to display in a button when the button is in its down state.

x-offset

An integer expression that specifies the pixel along the x-axis at which to begin reading from the image file.

y-offset

An integer expression that specifies the pixel along the y-axis at which to begin reading from the image file.

width

An integer expression that specifies the number of pixels along the x-axis to read from the image file.

height

An integer expression that specifies the number of pixels along the y-axis to read from the image file.

The image is not displayed until the widget is realized. If the read is successful, the method returns TRUE.

In Windows, you can specify a URL pathname. If you specify a fully-qualified URL, LOAD-IMAGE-DOWN( ) loads the image file directly without searching directories or URLs in PROPATH. Valid URL protocols include HTTP and HTTPS.

Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, the LOAD-IMAGE-DOWN( ) method continues searching with the next PROPATH entry.

If you specify URL pathnames on the PROPATH and your application repeatedly uses the LOAD-IMAGE-DOWN( ) method with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the image files. Use this value to create a fully-qualified URL pathname for *filename* and avoid repeated searches of the PROPATH.

LOAD-IMAGE-INSENSITIVE( ) method
(Windows only; Graphical interfaces only)

Reads the image contained in the specified file. The image is used for the button in its insensitive state.

Return type: LOGICAL

Applies to: BUTTON widget

Syntax

```
LOAD-IMAGE-INSENSITIVE ( filename
   [ , x-offset , y-offset , width , height ] )
```

filename

A character-string expression that specifies a full or relative pathname for a file that contains an image to display in a button when the button is insensitive.

x-offset

An integer expression that specifies the pixel along the x-axis at which to begin reading from the image file.

y-offset

An integer expression that specifies the pixel along the y-axis at which to begin reading from the image file.

width

An integer expression that specifies the number of pixels along the x-axis to read from the image file.

height

An integer expression that specifies the number of pixels along the y-axis to read from the image file.

The image is not displayed until the widget is realized. If the read is successful, the method returns TRUE. When the button becomes insensitive, and if the button does not have an INSENSITIVE-IMAGE, the AVM automatically grays out the button's UP-IMAGE.

In Windows, you can specify a URL pathname. If you specify a fully-qualified URL, LOAD-IMAGE-INSENSITIVE( ) loads the image file directly without searching directories or URLs in PROPATH. Valid URL protocols include HTTP and HTTPS.

Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, the LOAD-IMAGE-INSENSITIVE( ) method continues searching with the next PROPATH entry.

If you specify URL pathnames on the PROPATH and your application repeatedly uses the LOAD-IMAGE-INSENSITIVE( ) method with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the image files. Use this value to create a fully-qualified URL pathname for *filename* and avoid repeated searches of the PROPATH.

LOAD-IMAGE-UP( ) method
(Windows only; Graphical interfaces only)

Reads the image contained in a specified file. The image is used for the button in its up state. The image is also used for the down state if a separate down image is not specified. This method is equivalent to the LOAD-IMAGE( ) method.

Return type: LOGICAL

Applies to: BUTTON widget

Syntax

```
LOAD-IMAGE-UP ( filename [ , x-offset , y-offset , width , height ] )
```

filename

A character-string expression that specifies a full or relative pathname for a file that contains an image to display in a button when the button is in its up state.

x-offset

An integer expression that specifies the pixel along the x-axis at which to begin reading from the image file.

y-offset

An integer expression that specifies the pixel along the y-axis at which to begin reading from the image file.

width

An integer expression that specifies the number of pixels along the x-axis to read from the image file.

height

An integer expression that specifies the number of pixels along the y-axis to read from the image file.

The image is not displayed until the button is realized. If the read is successful, the method returns TRUE.

In Windows, you can specify a URL pathname. If you specify a fully-qualified URL, LOAD-IMAGE-UP( ) loads the image file directly without searching directories or URLs in PROPATH. Valid URL protocols include HTTP and HTTPS.

Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, the LOAD-IMAGE-UP( ) method continues searching with the next PROPATH entry.

If you specify URL pathnames on the PROPATH and your application repeatedly uses the LOAD-IMAGE-UP( ) method with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the image files. Use this value to create a fully-qualified URL pathname for *filename* and avoid repeated searches of the PROPATH.

LOAD-MOUSE-POINTER( ) method
(Windows only; Graphical interfaces only)

Specifies the mouse pointer to display when the pointer is moved over the widget. If you apply this method to a frame, field group, or window, the same mouse pointer is displayed when it is moved across all child widgets within the frame, field group, or window. However, if you load a different mouse pointer for a child widget, the child widget mouse pointer is displayed when it is moved over that child.

Return type: LOGICAL

Applies to: BROWSE widget (browse and column), BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

Syntax

```
LOAD-MOUSE-POINTER ( pointer-name )
```

pointer-name

A character-string expression that specifies the name of a mouse pointer. The expression can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

ABL provides a collection of mouse pointers that you can use in graphical applications. The following table names and describes each mouse pointer in the collection.

Table 85: ABL mouse pointers

Pointer name	Description
APPSTARTING	Arrow with an hourglass beside it
ARROW	Standard arrow cursor
CROSS	Cross hairs
HELP	Arrow with a question mark beside it
IBEAM	I-beam text cursor
NO	Circle with a slash through it
RECTANGLE	(NT 3.51 only) White rectangle
SIZE	Sizing rectangle
SIZE-E	Size to right
SIZE-N	Size to top
SIZE-NE	Size to top right
SIZE-NW	Size to top left
SIZE-S	Size to bottom
SIZE-SE	Size to bottom right
SIZE-SW	Size to bottom left
SIZE-W	Size to left
UPARROW	Up arrow
WAIT	System busy
GLOVE	Glove/finger
COMPILER-WAIT	Compiler busy

If the mouse pointer is loaded successfully, the method returns TRUE.

In addition to the mouse pointers that ABL supplies, you can also use a bitmap that you supply that is in the form of a Windows cursor (`.cur` or `.ani`) file. To use such a bitmap, substitute the name of the Windows cursor file for *`pointer-name`*.

For browse columns, if you do not specify a mouse pointer, the AVM uses the mouse pointer the user specified for the browse.

In Windows, you can specify a URL pathname. If you specify a fully-qualified URL, LOAD-MOUSE-POINTER( ) loads the pointer file directly without searching directories or URLs in PROPATH. Valid URL protocols include HTTP and HTTPS.

Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, the LOAD-MOUSE-POINTER( ) method continues searching with the next PROPATH entry.

If you specify URL pathnames on the PROPATH and your application repeatedly uses the LOAD-MOUSE-POINTER( ) method with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the pointer files. Use this value to create a fully-qualified URL pathname for *pointer-name* and avoid repeated searches of the PROPATH.

LOAD-SMALL-ICON( ) method
(Windows only; Graphical interfaces only)

Loads the icon you want from a specified file for display in the title bar of a window and in the task bar only. This method can accommodate icons formatted as small size (16x16) icons, regular size (32x32) icons, or both.

The icon file might contain multiple icons. In those instances when multiple icons are in a file, the LOAD-SMALL-ICON( ) method, by default, uses the 16x16 icon, if one exists, from the file that you specified. Otherwise, it uses the first icon in the file. If it uses a 32x32 icon, it reduces its size to a 16x16 format in both the title bar and the task bar.

If the load is successful, this method returns TRUE.

Note: You cannot use this method to display a specific icon when selecting a program using **ALT+TAB**.

Return type: LOGICAL

Applies to: WINDOW widget

Syntax

```
LOAD-SMALL-ICON ( smallicon-filename [ n ] )
```

smallicon-filename

A character-string expression that specifies the name of a file that contains the icon you want to load for display in the title bar of a window and in the task bar.

In Windows, you can specify a URL pathname for *smallicon-filename*. If you specify a fully-qualified URL, LOAD-SMALL-ICON( ) loads the icon file directly without searching directories or URLs in PROPATH. Valid URL protocols include HTTP and HTTPS.

Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, the LOAD-SMALL-ICON( ) method continues searching with the next PROPATH entry.

If you specify URL pathnames on the PROPATH and your application repeatedly uses the LOAD-SMALL-ICON( ) method with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the icon files. Use this value to create a fully-qualified URL pathname for *smallicon-filename* and avoid repeated searches of the PROPATH.

If you specify an empty string ("") for *smallicon-filename*, this method removes the most recently loaded icon.

The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

n

An integer expression that specifies the position of an icon within the file. Only use this expression if you want to override the default behavior.

For example, the following invocation of LOAD-ICON( ) finds the second icon in the icon file `file.ico` and loads it:

```
LOAD-ICON("file.ico", 2).
```

The following invocation removes the previously loaded icon:

```
LOAD-ICON("").
```

The LOAD-SMALL-ICON( ) method is only available in Windows. If you try to use it with any other platform, this method returns FALSE.

You typically use this method to load a small icon for an ABL window to display in its title bar or on the taskbar. ABL has no means of specifying a default icon for this purpose, and Windows provides a generic icon if you do not specify one.

.NET forms allow you to specify a default icon that works for this purpose using the form Icon property, which in ABL is provided by the Progress.Windows.Form class. You can then change this property as required for different uses.

You can maintain uniform icons across .NET forms and ABL windows in an application by using this method to load the same small icon you are using to set the Icon property for .NET forms.

See also

LOAD-ICON( ) method

LOCAL-HOST attribute

Indicates the IP (Internet Protocol) address of the machine the socket object is communicating with.

Data type: CHARACTER

Access: Read-only

Applies to: Socket object handle

When a server and client successfully establish a connection, both the server and client have a socket object that identifies this connection. On the client and on the server, this attribute returns the IP address of the machine which is making the request. If the CONNECT method failed, this attribute returns the Unknown value (?).

LOCAL-NAME attribute

This attribute returns the unqualified part of a namespace-aware XML node name or SOAP-header-entryref element name (that is, the part after the colon character). For nodes created with the CREATE-NODE( ) method, or nodes of any type other than ELEMENT or ATTRIBUTE, this attribute returns " ".

Data type: CHARACTER

Access: Read-only

Applies to: SOAP-header-entryref object handle, X-noderef object handle

LOCAL-PORT attribute

Indicates the port number of the socket.

Data type: INTEGER

Access: Read-only

Applies to: Socket object handle

When a server and client successfully establish a connection, both the server and client have a socket object that identifies this connection. On the client, this attribute returns the port number used on the client machine for this socket connection. On the server, this attribute returns the port number used on the server machine for this socket connection. If the CONNECT failed, this attribute returns the Unknown value (?).

LOCAL-VERSION-INFO attribute

An object reference to a `Progreses.Lang.OEVersionInfo` class that provides version information for the current instance of the OpenEdge ABL Virtual Machine (AVM). If an ABL client accesses this attribute, it describes the OpenEdge version of the client. If an application server session accesses this attribute, it describes the OpenEdge version of the application server session, and so on.

Data type: Progress.Lang.OEVersionInfo class

Access: Read-only

Applies to: SESSION system handle

See also

CURRENT-REQUEST-INFO attribute, CURRENT-RESPONSE-INFO attribute, REQUEST-INFO attribute, RESPONSE-INFO attribute, VersionInfo property

LOCATOR-COLUMN-NUMBER attribute

The current column in the XML source.

Data type: INTEGER

Access: Read-only

Applies to: SAX-reader object handle

Valid only in a callback. Use the SELF handle to get it, as in the following fragment:

```
myColNum = SELF:LOCATOR-COLUMN-NUMBER.
```

Gives the column number where the text that caused the current callback ends. The first column in a line is 1. If the current location is an **external entity**—that is, external to the main XML source—the column number is relative to the beginning of the line in the external entity.

LOCATOR-LINE-NUMBER attribute

The current line in the XML source.

Data type: INTEGER

Access: Read-only

Applies to: SAX-reader object handle

Valid only in a callback. Use the SELF handle to get it, as in the following fragment:

```
myLineNum = SELF:LOCATOR-LINE-NUMBER.
```

Gives the line number where the text that caused the current callback ends. The first line in a document is 1. If the current location is an **external entity**—that is, external to the main XML source—the line number is relative to the beginning of the external entity.

LOCATOR-PUBLIC-ID attribute

Returns the public identifier of the current XML source.

Data type: CHARACTER

Access: Read-only

Applies to: SAX-reader object handle

Valid only in a callback. Use the SELF handle to get the public identifier of the XML source, as in the following fragment:

```
myPublicID = SELF:LOCATOR-PUBLIC-ID.
```

LOCATOR-SYSTEM-ID attribute

Returns the system identifier of the current XML source.

Data type: CHARACTER

Access: Read-only

Applies to: SAX-reader object handle

Valid only in a callback. Use the SELF handle to get the system identifier of the XML source, as in the following fragment:

```
mySystemID = SELF:LOCATOR-SYSTEM-ID.
```

LOCATOR-TYPE attribute

The type of server on which the application files are stored.

Data type: CHARACTER

Access: Read-only

Applies to: CODEBASE-LOCATOR system handle

Valid values are "AppServer" or "InternetServer".

LOCKED attribute

Indicates whether another user has a lock on a record that a FIND . . . NO-WAIT or GET . . . NO-WAIT statement or method is trying to access.

Data type: LOGICAL

Access: Read-only

Applies to: Buffer object handle

Note: The LOCKED attribute corresponds to the LOCKED function.

LOCK-REGISTRATION( ) method

Prevents the registration of additional domains in the ABL session domain registry for the remainder of an ABL session. You must call this method to use the domains you have registered in the session domain registry using the REGISTER-DOMAIN( ) method.

Note: This method is not supported for any session domain registry with which you intend to authenticate or validate (through single sign-on) user identity for a multi-tenant database connection.

Return type: LOGICAL

Applies to: SECURITY-POLICY system handle

Syntax

```
LOCK-REGISTRATION( )
```

You must call this method before you can use a session domain registry built using the REGISTER-DOMAIN( ) method to authenticate or validate (through a single sign-on operation) the user identity represented by a client-principal object in the ABL session.

If you do not register at least one domain in the session domain registry before calling this method, this method returns TRUE. However, any attempt to seal a client-principal object raises a run-time error.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

Note: Once you call this method in a session, you can no longer call the REGISTER-DOMAIN( ) method or the LOAD-DOMAINS( ) method. Until you call this method, you can continue to build the session registry using calls to the REGISTER-DOMAIN( ) method or reload the registry from the specified database multiple times in an ABL session using calls to LOAD-DOMAINS( ).

See also

LOAD-DOMAINS( ) method, REGISTER-DOMAIN( ) method

LOG-AUDIT-EVENT( ) method

Creates an audit record for the specified application-defined audit event in each connected audit-enabled database whose current audit policy has this audit event enabled.

This method returns a Base64 character string that specifies a universally unique identifier (UUID) as the primary index for the generated audit event record. The UUID is 22 characters in length (the two trailing Base64 pad characters are removed).

Return type: CHARACTER

Applies to: AUDIT-CONTROL system handle

Syntax

```
LOG-AUDIT-EVENT ( event-id , event-context [ , event-detail
  [ , audit-custom-detail ] ] )
```

event-id

An integer value that specifies an identifier for an application-defined audit event. This value must be greater than or equal to 32000.

event-context

A character expression that specifies the context for the audit event. The value of this expression cannot exceed 200 characters. You can also use this value as an alternate index for querying the audit event record.

If you specify the Unknown value (?), the AVM generates a run-time error.

event-detail

An optional character expression that specifies additional audit detail. The value of this expression cannot exceed 10,000 characters.

audit-custom-detail

An optional character expression that specifies additional user detail. The value of this expression cannot exceed 10,000 characters.

LOG-ENTRY-TYPES attribute

A comma-separated list of one or more types of log entries to write to the log file.

Data type: CHARACTER

Access: Readable/Writable for LOG-MANAGER, Read-only for DSLOG-MANAGER

Applies to: DSLOG-MANAGER system handle, LOG-MANAGER system handle

Use the LOG-ENTRY-TYPES attribute to specify one or more types of log entries to write to the log file. For an OpenEdge client, the Client Logging (`-clientlog`) startup parameter or the LOG-MANAGER system handle's LOGFILE-NAME attribute specifies the log file. For a DataServer server, the DataServer Logging (`-dslog`) startup parameter specifies the log file.

The LOG-ENTRY-TYPES attribute corresponds to the Log Entry Types (`-logentrytypes`) startup parameter.

By default, the logging level you specify using the LOGGING-LEVEL attribute or the Logging Level (`-logginglevel`) startup parameter applies to all log entry types specified. However, you can specify a different logging level for each entry type by assigning a character expression as follows:

Syntax

```
LOG-MANAGER:LOG-ENTRY-TYPES =
  "log-entry-type[:level][,log-entry-type[:level]]..."
```

log-entry-type

A log entry type listed in the following tables.

level

A logging level value (between 0 and 4).

The following table describes the log entry types for the LOG-MANAGER system handle.

Table 86: LOG-MANAGER system handle log entry types

Log entry type	Executables	Description
4GLMessages	ABL (GUI and character mode). Application servers do not require this log entry type for ABL messages to be written to the log file	Turns on logging of ABL messages. OpenEdge writes all ABL VIEW-AS ALERT-BOX messages to the log file, together with the ABL stack, when you turn on Debug Alert using either the Debug Alert (-debugalert) startup parameter or the DEBUG-ALERT attribute on the SESSION system handle.
4GLTrace	ABL clients, application servers	Turns on logging for the execution of internal procedures, user-defined functions, persistent user-interface triggers, named events, and class events (generated by the RUN, FUNCTION, PUBLISH, and SUBSCRIBE statements, respectively, and for class events; by the Publish( ), Subscribe( ), and Unsubscribe( )event methods, respectively). It also logs the instantiation and use of classes, including execution of constructors (invoked by the NEW function (classes) and also by the SUPER and THIS-OBJECT statements), the execution of methods defined within classes (including those invoked using the SUPER system reference), the execution of property accessors (invoked by accessing a property of a class), and the execution of destructors (invoked by executing the DELETE OBJECT statement or as a result of ABL garbage collection).
4GLTrans	ABL clients	Turns on logging for the processing of transactions and subtransactions in ABL procedures.
ASDefault	Application servers	Combines the ASPlumbing and DB.Connects log entry types. It is the default value for application server sessions.
ASPlumbing	Application servers	Turns on logging for different actions, depending on the logging level specified.

Log entry type	Executables	Description
DB.Connects	ABL clients, application servers	Turns on logging of database connections (connects and disconnects). The log messages include database name and user ID number.
DS.Cursor DS.QryInfo	ABL clients	Turns on logging of DataServer processing
DS.Performance	DataServer clients	Turns on logging for the client-side performance details.
DynObjects.Class DynObjects.DB DynObjects.XML DynObjects.Other DynObjects.UI	ABL clients, application servers	Turns on logging of dynamic object creation and deletion. For a list of objects within each category and a description of the log entries, see *Troubleshoot ABL Applications*.
FileID	ABL clients, application servers	Turns on logging of file access operations (opening and closing files), and any file access error messages that might occur.
IgnoredOps	ABL clients	Turns on logging of operations that the AVM ignores because the ABL windows to which they apply are embedded in .NET forms. For information about operations that the AVM ignores when they occur in the context of an embedded window, see EmbeddedWindow property.
ProEvents.UI.Char ProEvents.UI.Command ProEvents.Other	ABL clients, application servers	Turns on event logging for different categories of events. For a list of events within each category and a description of the log entries, see *Troubleshoot ABL Applications*.
QryInfo	ABL clients, application servers	Turns on logging of queries (each open query and FOR EACH block) executed in an application.
SAX	ABL clients, application servers	Turns on logging for the SAX parser.
Sec.TenancyTrace	Application servers	Turns on logging for an application's access to tenant data deployed in an OpenEdge database multi-tenant environment.

The following table describes the log entry types for the DSLOG-MANAGER system handle.

Table 87: DSLOG-MANAGER log entry types

Log entry type	Description
`Connects`	Turns on logging for connection attributes and related information in the DataServer log file.
`Cursor`	Turns on logging of cursor details in the DataServer log file.
`SQL`	Turns on logging of SQL query processing in the DataServer log file.
`Trans`	Turns on logging for the processing of transactions and subtransactions in the DataServer log file.
`Performance`	Turns on logging of performance-related information in the server context.

The following example shows how to specify one or more log entry types:

```
LOG-MANAGER:LOG-ENTRY-TYPES = "DB.Connects,4GLTrace:2,DynObjects.UI:3"
```

Note: If you set the LOGGING-LEVEL attribute at runtime, it resets the logging level for all log entry types to the specified value. You should use LOG-ENTRY-TYPES to change logging levels for specific log entry types at runtime. You must re-specify all the different log entry types each time that you want to change one.

The following example shows how to specify all log entry types within a category:

```
LOG-MANAGER:LOG-ENTRY-TYPES = "DynObjects.*"
```

You can also turn off logging by setting this attribute to the Unknown value (?).

See also the reference entries for the Client Logging (`-clientlog`), DataServer Logging (`-dslog`), Log Entry Types (`-logentrytypes`), and Logging Level (`-logginglevel`) startup parameters in *Startup Command and Parameter Reference*.

For more information about log entry types and logging levels, see *Troubleshoot ABL Applications*.

LOG-THRESHOLD attribute

The file size threshold of log files. When the current log file becomes equal to or greater than the specified size, the AVM renames and saves the log file and creates a new log file.

Data type: INTEGER

Access: Read-only

Applies to: DSLOG-MANAGER system handle, LOG-MANAGER system handle

Valid values are:

- **0** — This means there is no limit other than what the operating system imposes. Specify 0 to ignore the Number of Log Files to Keep (`-numlogfiles`) startup parameter setting. This is the default.
- **Between 500,000 and 2,147,483,647** — Values are in bytes (one byte typically holds one character). You can specify a file size up to 2GB, inclusive, but not lower than 500,000.

The AVM names log files controlled by either the `DSLOG-MANAGER` or `LOG-MANAGER` system handles based on a sequence number using the following format:

Syntax

```
filename.999999.extension
```

For example, if you specify a log file named `my.log`, the AVM renames the log file to `my.000001.log` before creating a new log file.

For the `LOG-MANAGER` system handle, the `LOG-THRESHOLD` attribute corresponds to the Log Threshold (`-logthreshold`) startup parameter. For the `DSLOG-MANAGER` system handle, this attribute corresponds to the Log Threshold (`-logthreshold`) startup parameter when the client is self-service or client/server using ProBroker.

To specify a log file name for ABL clients, use the LOGFILE-NAME attribute of the `LOG-MANAGER` system handle or the Client Logging (`-clientlog`) startup parameter. Use the NUM-LOG-FILES attribute of the `LOG-MANAGER` system handle or the Number of Log Files to Keep (`-numlogfiles`) startup parameter to specify the number of log files to keep.

For DataServers, use the DataServer Logging (`-dslog`) startup parameter to specify a log file name.

Note: The `LOGFILE-NAME` attribute of the `DSLOG-MANAGER` system handle is read-only.

For more information about the Client Logging (`-clientlog`), DataServer Logging (`-dslog`), Log Threshold (`-logthreshold`), and Number of Log Files to Keep (`-numlogfiles`) startup parameters, see *Startup Command and Parameter Reference*, *Use the Microsoft SQL Data Server*, and *Troubleshoot ABL Applications*.

LOGFILE-NAME attribute

The name of log file OpenEdge uses to log messages and ABL stack trace information.

Data type: CHARACTER

Access: Readable/Writeable for `LOG-MANAGER`, Read-only for `DSLOG-MANAGER`

Applies to: DSLOG-MANAGER system handle, LOG-MANAGER system handle

If the filename you supply is a relative pathname, then a file is accessed relative to the current working directory. If the filename is an absolute pathname, then the specified file is accessed. In Windows, the pathname cannot contain characters outside of the non-Unicode code page. See *Internationalize ABL Applications* for more information about Unicode and code pages.

Note: Do not include a numbered sequence in the filename. This might conflict with the rolled over log files OpenEdge creates based on your NUM-LOG-FILES attribute and LOG-THRESHOLD attribute settings.

When the specified log file is open, this `LOG-MANAGER` attribute is read-only. This attribute is always read-only for the `DSLOG-MANAGER` system handle.

For the LOG-MANAGER system handle, the LOGFILE-NAME attribute corresponds to the Client Logging (-clientlog) startup parameter. For the DSLOG-MANAGER system handle, this attribute corresponds to the DataServer Logging (-dslog) startup parameter when the client is self-service or client/server using ProBroker.

Use the LOG-ENTRY-TYPES attribute or the Log Entry Types (-logentrytypes) startup parameter to specify one or more types of log entries you want to write to the log file. Use the LOGGING-LEVEL attribute or the Logging Level (-logginglevel) startup parameter to specify a general level at which log entries are written to the log file.

For more information about the Client Logging (-clientlog), DataServer Logging (-dslog), Log Entry Types (-logentrytypes), and Logging Level (-logginglevel) startup parameters, see *Startup Command and Parameter Reference*, *Use the Microsoft SQL Data Server*, and *Troubleshoot ABL Applications*.

LOGGING-LEVEL attribute

The level at which log entries are written to the log file.

Data type: INTEGER

Access: Readable/Writable

Applies to: DSLOG-MANAGER system handle, LOG-MANAGER system handle

Use the LOGGING-LEVEL attribute to specify the level at which log entries are written to the log file. For an OpenEdge client, the Client Logging (-clientlog) startup parameter or the LOG-MANAGER system handle's LOGFILE-NAME attribute specifies the log file. For a DataServer server, the DataServer Logging (-dslog) startup parameter specifies the log file. Each logging level specifies a different amount of information.

There are five logging levels:

Logging level	Description
0 (None)	Logs no entries. This is equivalent to turning logging off.
1 (Errors)	Logs OpenEdge error messages. This includes all error messages and is unrelated to the entry types specified. Errors continue to be logged at all higher levels.
2 (Basic)	The log entry type determines what information is logged. Each log entry type generates at least some output. This is the default.
3 (Verbose)	The log entry type determines what information is logged. Includes the information generated at levels 1 and 2.
4 (Extended)	The log entry type determines what information is logged.

By default, the logging level you specify applies to all log entry types. However, you can specify a different logging level for individual log entry types with the LOG-ENTRY-TYPES attribute or the Log Entry Types (-logentrytypes) startup parameter. The higher logging levels include the log information from the lower logging levels, in addition to the information logged at that level.

Note: If you set the LOGGING-LEVEL attribute at runtime, it resets the logging level for all log entry types to the specified value. You should use LOG-ENTRY-TYPES to change logging levels for specific log entry types at runtime. You must re-specify all the different log entry types each time that you want to change one.

The LOGGING-LEVEL attribute corresponds to the Logging Level (-logginglevel) startup parameter.

For more information about the Client Logging (`-clientlog`), DataServer Logging (`-dslog`), Log Entry Types (`-logentrytypes`), or Logging Level (`-logginglevel`) startup parameters, see *Startup Command and Parameter Reference*, *Use the Microsoft SQL Data Server*, *Use the Oracle Data Server*, and *Troubleshoot ABL Applications*.

LOGIN-EXPIRATION-TIMESTAMP attribute

The time stamp specifying when the client-principal object will expire. If the client-principal object expires before you can seal or validate it, and you attempt to use it, the AVM sets the LOGIN-STATE attribute to `"EXPIRED"` and you can no longer validate or use the client-principal object.

Note: If you set the attribute to the Unknown value (?), the client-principal is reset to have no expiration.

Data type: DATETIME-TZ

Access: Readable/Writeable

Applies to: Client-principal object handle

You can set this attribute to any DATETIME-TZ value, with the following limitations. The value of this attribute:

- Is truncated to the second
- Can have no greater precision than seconds
- Can have a minimum value of `1/1/1970 00:00:01.000-00:00`
- On 32-bit platforms, can have the following maximum value: `1/19/2038 00:00:00.000-00:00`
- On 64-bit platforms, has no meaningful maximum value

Any attempt to set a value that does not conform to these limitations returns a message in the ERROR-STATUS system handle and leaves the previous value unchanged.

If not set or reset to the Unknown value (?), the AVM will never place the client-principal object in an EXPIRED login state. In either case, reading the attribute returns the Unknown value (?).

Note: The AVM recognizes that a client-principal object has expired only when it tries to use it with the SEAL( ) method, SECURITY-POLICY:SET-CLIENT( ) method, SET-DB-CLIENT function, IMPORT-PRINCIPAL( ) method, or VALIDATE-SEAL( ) method.

Once the client-principal object is sealed, this attribute is read-only.

See also

LOGIN-STATE attribute

LOGIN-HOST attribute

The name of the host system on which the user represented by the client-principal object was authenticated. If not specified, the AVM returns a zero-length character string.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

Once the client-principal object is sealed, this attribute is read-only.

LOGIN-STATE attribute

Returns a character value that represents the current state of the client-principal object.

Data type: CHARACTER

Access: Read-only

Applies to: Client-principal object handle

The following table lists the valid values and how they can be set.

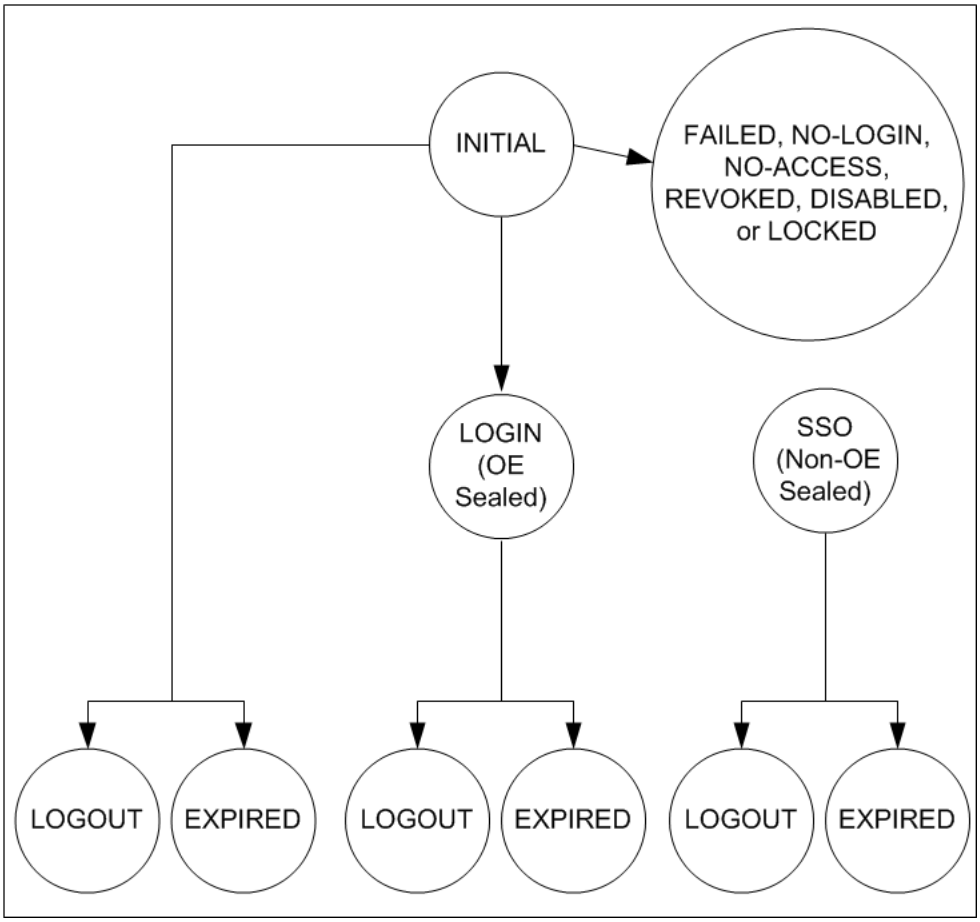
Table 88: Client-principal LOGIN-STATE attribute values

This value...	Is set...
"INITIAL"	(Default value) From the creation of an unsealed client-principal object using the CREATE CLIENT-PRINCIPAL until the object is sealed, and after calling the INITIALIZE( ) on a sealed or unsealed client-principal object in any state.
"LOGIN"	After the user identity represented by an unsealed client-principal object is successfully authenticated and the object is sealed using the SEAL( ) method, SECURITY-POLICY:SET-CLIENT( ) method, the SET-DB-CLIENT function, or via an OpenEdge STS.
"SSO"	After a client principal is sealed by a non-OpenEdge user authentication process.
"LOGOUT"	After executing the LOGOUT( ) method on a sealed client-principal object in the LOGIN state or SSO state.
"EXPIRED"	After executing the SEAL( ) method, SET-CLIENT( ) method, SET-DB-CLIENT function, or IMPORT-PRINCIPAL( ) method, VALIDATE-SEAL( ) method, or some external, non-OpenEdge seal procedure and the date and time set for the LOGIN-EXPIRATION-TIMESTAMP attribute has expired.
"FAILED"	After executing the AUTHENTICATION-FAILED( ) method on an unsealed client-principal object or after a user authentication operation fails on an unsealed client-principal object when calling the SET-CLIENT( ) method or SET-DB-CLIENT function.
"NO-LOGIN"	The login operation could not be completed. This state is set after a user authentication operation on an unsealed client-principal object fails due to some policy restriction when calling the SET-CLIENT( ) method or SET-DB-CLIENT function, where the authentication is done by an OpenEdge STS.
"NO-ACCESS"	The user is denied access. This state is set after a user authentication operation on an unsealed client-principal object fails due to some policy restriction when calling the SET-CLIENT( ) method or SET-DB-CLIENT function, where the authentication is done by an OpenEdge STS.

This value...	Is set...
"REVOKED"	The user account's access has been revoked. This state is set after a user authentication operation on an unsealed client-principal object fails due to some policy restriction when calling the SET-CLIENT( ) method or SET-DB-CLIENT function, where the authentication is done by an OpenEdge STS.
"DISABLED"	The user account is disabled. This state is set after a user authentication operation on an unsealed client-principal object fails due to some policy restriction when calling the SET-CLIENT( ) method or SET-DB-CLIENT function, where the authentication is done by an OpenEdge STS.
"LOCKED"	The user account is locked. This state is set after a user authentication operation on an unsealed client-principal object fails due to some policy restriction when calling the SET-CLIENT( ) method or SET-DB-CLIENT function, where the authentication is done by an OpenEdge STS.

The following figure illustrates the supported transitions between the states described in the table above.

Figure 2: Valid changes to LOGIN-STATE attribute value



When the AVM sets this attribute, it also sets the STATE-DETAIL attribute with a description of the current state. If this setting occurs as a result of invoking the AUTHENTICATION-FAILED( ) method, the AVM uses any string value you pass to this method to set the STATE-DETAIL attribute.

Attempting to write to this attribute returns a warning message.

See also

AUTHENTICATION-FAILED( ) method, CREATE CLIENT-PRINCIPAL statement, IMPORT-PRINCIPAL( ) method, INITIALIZE( ) method, LOGIN-EXPIRATION-TIMESTAMP attribute, LOGOUT( ) method, SEAL( ) method, SET-CLIENT( ) method, SET-DB-CLIENT function, STATE-DETAIL attribute, VALIDATE-SEAL( ) method

---

LOGOUT( ) method

Indicates that the user represented by the sealed client-principal object (in the LOGIN state) has logged out of their current user login session. This is a general purpose method an application can use to invalidate, or terminate access to, a sealed client-principal object. Once logged out, you can no longer use the client-principal object to set a user identity using either the SECURITY-POLICY:SET-CLIENT( ) method or SET-DB-CLIENT function.

Return type: LOGICAL

Applies to: Client-principal object handle

Syntax

```
LOGOUT( )
```

If you call this method for an unsealed client-principal object, the AVM generates a run-time error. You can use the AUTHENTICATION-FAILED( ) method to invalidate an **unsealed** client-principal object.

If successful, this method returns TRUE. Otherwise, it returns FALSE. The AVM also sets the LOGIN-STATE attribute for the client-principal object to `"LOGOUT"`.

Calling this method generates an audit event and creates an audit record for the event in all connected audit-enabled databases according to each database's current audit policy settings.

See also

AUTHENTICATION-FAILED( ) method, LOGIN-STATE attribute, SET-CLIENT( ) method, SET-DB-CLIENT function

---

LONGCHAR-TO-NODE-VALUE( ) method

Sets the value of an X-noderef node to the contents of a LONGCHAR.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
LONGCHAR-TO-NODE-VALUE ( longchar )
```

longchar

An expression of type LONGCHAR.

LONGCHAR-TO-NODE-VALUE( ) raises an error if any of the following occurs:

- The node is read only.
- The node contains invalid XML characters.
- The data in *longchar* is not null-terminated.
- *longchar* is not set to the exact size of the valid data.

For more information on accessing XML documents using the Document Object Model (DOM) interface, see *Use XML with ABL Applications*.

LOOKUP( ) method

Returns the index of the specified item in a combo-box list or selection list.

Return type: INTEGER

Applies to: BROWSE widget (column), COMBO-BOX widget, SELECTION-LIST widget

Syntax

```
LOOKUP ( list-string )
```

list-string

A character-string expression that specifies a single value in the combo box or selection list.

If *list-string* has the Unknown value (?), LOOKUP returns the Unknown value (?). If *list-string* is not in the list, LOOKUP returns 0.

For browses, this method applies only to combo-box browse columns.

MANDATORY attribute

Indicates whether a buffer-field is a required field.

Data type: LOGICAL

Access: Read-only

Applies to: Buffer-field object handle

MANUAL-HIGHLIGHT attribute

Indicates whether a widget exhibits custom or standard highlight behavior when selected.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

Set the MANUAL-HIGHLIGHT attribute to TRUE to use a customized highlight design for selection of the widget. A FALSE value for this attribute specifies the ABL default highlight behavior for the selection of the widget.

MARK-NEW( ) method

For a table with only new data, creates a blank before-table record for each buffer record, with a ROW-STATE of ROW-CREATED.

Return type: LOGICAL

Applies to: Buffer object handle

If a ProDataSet temp-table is known to contain only new data, MARK-NEW( ) can be used to create a before-table record for those buffer records that do not have one. ROW-STATE is set to ROW-CREATED. Using this method is easier than using MARK-ROW-STATE(ROW-CREATED), because you do not have to manually loop through all the records in the table.

MARK-NEW( ) does not support record deletes or updates.

If buffer object handle points to a static temp-table that was defined without a BEFORE-TABLE option, MARK-NEW( ) causes the AVM to raise ERROR. If buffer object handle is for a dynamic temp-table that has no before-table, one is created on-the-fly.

See also

AFTER-TABLE attribute, MARK-ROW-STATE( ) method

MARK-ROW-STATE( ) method

Forces the creation of before-table records and assigns ROW-STATE.

Return type: LOGICAL

Applies to: Buffer object handle

If a ProDataSet contains changes to be applied to a data-source, but no BEFORE-TABLE records exist for it and therefore no ROW-STATE, you may still want to use the SAVE-ROW-CHANGES( ) method to move the new data from the ProDataSet to the source database. However, this is problematic, since SAVE-ROW-CHANGES( ) requires a before-table record with a ROW-STATE for each record to be saved. This method creates the before-table records needed by SAVE-ROW-CHANGES( ).

Syntax

```
MARK-ROW-STATE ( row-state [ , before-image-source-buffer-handle ] )
```

row-state

An integer expression evaluating to one of the row-state values or one of the compiler constants ROW-CREATED, ROW-MODIFIED or ROW-DELETED shown in the following table.

Table 89: Row state values

Compiler constant	Value	Description
ROW-UNMODIFIED	0	The row was not modified.
ROW-DELETED	1	The row was deleted.
ROW-MODIFIED	2	The row was modified.
ROW-CREATED	3	The row was created.

before-image-source-buffer-handle

An optional handle to the before-image source buffer.

If *row-state* is ROW-UNMODIFIED, or if there is already a before-table record, no action is taken.

If *row-state* is ROW-CREATED, a before-table record is created and its ROW-STATE is marked as ROW-CREATED. The before-table record is associated with the buffer object handle record.

If *row-state* is ROW-MODIFIED and the second optional parameter is defined, a before-image record is created by copying the before-image source buffer. If the second parameter is not defined, then a data-source must be attached to the buffer object handle. In this case the data-source mapping is used to locate the data-source record(s) related to the buffer object handle, which is used to create a before-table record. The before-table record is marked ROW-MODIFIED and is associated with the buffer object handle record.

Note: If the second parameter is not defined and if the data-source top level record cannot be located, then a ROW-CREATED before-table record will be created and associated with the buffer object handle record.

If *row-state* is ROW-DELETED and the second optional parameter is defined, a before-image record is created by copying the before-image source buffer. If the second parameter is not defined, then a data-source must be attached to buffer object handle. In this case the data-source mapping is used to locate the data-source record(s) related to buffer object handle, which then can be used to create a before-table record.

If a second parameter is not defined, then the buffer object handle must contain a temporary record whose only purpose is to locate the data-source. Otherwise it can be empty.

Note that if buffer object handle is for a static table that was defined without a BEFORE-TABLE option, MARK-ROW-STATE( ) causes the AVM to raise ERROR. If buffer object handle is for a dynamic table that has no before-table, one is automatically created.

See also

MARK-NEW( ) method, ROW-STATE attribute, SAVE-ROW-CHANGES( ) method

---

MAX-BUTTON attribute
(Windows only; Graphical interfaces only)

Determines whether the window has a maximize button in its caption bar.

Data type: LOGICAL

Access: Readable/Writable

Applies to: WINDOW widget

In character interfaces, this attribute has no effect.

In Windows, a window can have maximize and minimize buttons depending on the settings of the MAX-BUTTON and MIN-BUTTON attributes. Both buttons are created on the window. If you set the MAX-BUTTON to TRUE and the MIN-BUTTON to FALSE, only the maximize button is enabled; the minimize button is disabled.

The MAX-BUTTON attribute must be set before the window is realized. The default value is TRUE.

MAX-CHARS attribute
(Graphical interfaces only)

The maximum number of characters an editor or combo-box widget can hold.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget

For editor widgets, you can set this attribute only before the widget is realized. In Windows, the maximum value of MAX-CHARS is approximately 20K for the regular editor and over 64K for the large editor.

For SIMPLE and DROP-DOWN combo-box widgets, you can set this attribute before or after the widget is realized. If the value of MAX-CHARS for a combo-box widget is zero or the Unknown value (?), the default value is 255 characters. This attribute is ignored for DROP-DOWN-LIST combo-box widgets.

Note: In character interfaces, editors can grow until the AVM runs out of system resources.

For browses, this attribute applies only to combo-box browse columns.

MAX-DATA-GUESS attribute

The estimated number of records in a browse query.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget

Before enabling the browse widget, set this attribute to the exact or maximum number of records you expect in the query. A more accurate setting of this attribute allows for a smoother and more accurate change in vertical thumb height when the user scrolls through the query for the first time. As a user scrolls through the records, the system continuously updates the value of this attribute with a better guess for the number of records. After all records have been read, the MAX-DATA-GUESS value is automatically reset to the exact number for more accurate browsing. The default value is 100.

MAX-HEIGHT-CHARS attribute

The maximum height of the window, in character units.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: WINDOW widget

MAX-HEIGHT-PIXELS attribute

The maximum height of the window, in pixels.

Data type: INTEGER

Access: Readable/Writeable

Applies to: WINDOW widget

MAXIMUM-LEVEL attribute

Forces a recursive data-relation to stop at a specific number of iterations of a child buffer. This attribute causes a recursive ProDataSet FILL to stop when the number of iterations of the child buffer reaches the MAXIMUM-LEVEL.

Data type: INTEGER

Access: Readable/Writeable

Applies to: Data-relation object handle

See also

DEFINE DATASET statement, GET-RELATION( ) method, RECURSIVE attribute

MAX-VALUE attribute

The maximum value for a slider.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SLIDER widget

You can set this attribute only before the widget is realized.

MAX-WIDTH-CHARS attribute

The maximum width of a window, in character units.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: WINDOW widget

MAX-WIDTH-PIXELS attribute

The maximum width of a window, in pixels.

Data type: INTEGER

Access: Readable/Writeable

Applies to: WINDOW widget

MD5-VALUE attribute

Obsolete as of OpenEdge Release 12.0. Use the SIGNATURE-VALUE attribute instead.

Data type: CHARACTER

Access: Read-only

Applies to: RCODE-INFO handle

Returns the Unknown (?) value.

MEMPTR-TO-NODE-VALUE( ) method

Sets the value of an X-noderef node to the contents of a MEMPTR.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
MEMPTR-TO-NODE-VALUE ( memptr )
```

memptr

An expression of type MEMPTR.

MEMPTR-TO-NODE-VALUE( ) raises an error if any of the following occurs:

- The node is read only.

- The node contains invalid XML characters.
- The data in *memptr* is not null-terminated.
- *memptr* is not set to the exact size of the valid data.

For more information on accessing XML documents using the Document Object Model (DOM) interface, see *Use XML with ABL Applications*.

MENU-BAR attribute

The handle of a menu bar widget associated with a window.

Data type: HANDLE

Access: Readable/Writeable

Applies to: WINDOW widget

You can establish the menu bar for a window by assigning the MENU-BAR attribute.

MENU-KEY attribute

The accelerator key sequence that activates the pop-up menu for a widget.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

Any value you set must evaluate to a valid ABL key label, such as "a", "F1", or "ALT-SHIFT-F1".

MENU-MOUSE attribute
(Graphical interfaces only)

The mouse button on a three-button mouse that activates the pop-up menu for a widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

The following table lists each mouse button and the attribute value that specifies it as the pop-up menu button.

Table 90: Pop-up menu button

Mouse button	Attribute value
Left	1

Mouse button	Attribute value
Middle	2
Right	3

If you use a two-button mouse, setting this attribute to "2" makes it impossible to access the menu with your mouse. If you do not set this attribute, it returns the Unknown value (?).

MERGE-BY-FIELD attribute

Specifies whether the AVM merges changes on a field-by-field basis when saving changes from a ProDataSet temp-table buffer to the associated data source using the SAVE-ROW-CHANGES( ) method. The default value is TRUE.

Note: Merging a large number of changes from a ProDataSet object to the data source on a field-by-field basis is slower than saving changes buffer-by-buffer.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Data-source object handle

If TRUE, the AVM performs a field-to-field comparison to identify data conflicts. In this case, a data conflict exists when the same field in the ProDataSet buffer and its associated data source buffer has changed. A data conflict would not exist if a different field had changed in either buffer.

If FALSE, the AVM performs a buffer-to-buffer comparison to identify data conflicts. In this case, a data conflict exists when any field in the data source buffer has changed.

MERGE-CHANGES( ) method

Merges the changed rows of all temp-tables or a single temp-table in a source ProDataSet object loaded with the GET-CHANGES( ) method into the corresponding rows of all temp-tables or a single temp-table (respectively) in the original (target) ProDataSet object.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle (of the source object)

The syntax to invoke this method follows:

Syntax

```
change-handle:MERGE-CHANGES ( original-handle [ , copy-all-mode ] )
```

change-handle

A handle to the source ProDataSet object or to the buffer object of the source ProDataSet temp-table that contains the changed rows.

original-handle

A handle to the original (target) ProDataSet object or the buffer object of a single target ProDataSet temp-table to merge with the changed rows from the source object.

copy-all-mode

An optional logical expression, where TRUE indicates that the AVM merges **all** rows of the source after-image table.

When TRUE, the AVM merges all after-image table rows whether or not they contain changes. In this case, the temp-table in the original ProDataSet object must have a unique primary index that the AVM can use to find each corresponding row from the after-image table (since unchanged rows do not have a corresponding row in the before-image table). When a corresponding row is not found in the original ProDataSet object, the AVM creates a new row using the row from the after-image table. When FALSE, the AVM merges only after-image table rows that contain changes. The default value is FALSE.

Notes

- For a ProDataSet object handle, all modified tables in the ProDataSet object are merged. For a Buffer object handle, only the temp-table associated with that buffer is merged.
- If the ERROR attribute or REJECTED attribute for a changed table is TRUE, the MERGE-CHANGES( ) method backs out the changes. Otherwise, this method accepts the changes by copying the after-image table rows to the corresponding after-image table rows in the original ProDataSet temp-table. The AVM also sets the BEFORE-ROWID attribute of the row in the after-image table to the Unknown value (?), sets the ROW-STATE of the row in the after-image table to ROW-UNMODIFIED (0), and removes the before-image table row (if it has one).
- When executing the method, there is no guarantee as to the contents of the buffers involved after the method finishes executing.

MERGE-ROW-CHANGES( ) method

Merges a single changed row from a source ProDataSet object loaded with the GET-CHANGES( ) method into the corresponding row in the original (target) ProDataSet temp-table buffer.

Return type: LOGICAL

Applies to: Buffer object handle (of the source ProDataSet)

The syntax for invoking this method follows:

Syntax

```
change-handle:MERGE-ROW-CHANGES ( [original-handle[ , copy-all-mode]] )
```

change-handle

A handle to the buffer object of a before-image temp-table for a changed row, or to the buffer object of an after-image temp-table for an unchanged row, of the source ProDataSet.

original-handle

A handle to the buffer of the original (target) ProDataSet temp-table to merge with the changed row from the source ProDataSet.

The AVM uses *original-handle* only to match to the original table currently associated with *change-handle* (specified in the ORIGIN-HANDLE attribute). The current row in this table is ignored.

copy-all-mode

An optional logical expression, where TRUE indicates that the AVM **always** merges the row.

When TRUE, the AVM merges the after-image table row whether or not it contains changes. In this case, the temp-table in the original ProDataSet object must have a unique primary index that the AVM can use to find the corresponding row from the after-image table (since an unchanged row does not have a corresponding row in the before-image table). When a corresponding row is not found in the original ProDataSet object, the AVM creates a new row using the row from the after-image table. When FALSE, the AVM merges only an after-image table row that contains changes. The default value is FALSE.

If the ERROR attribute or REJECTED attribute for the changed row is TRUE, the MERGE-ROW-CHANGES( ) method backs out the change. Otherwise, this method accepts the change by copying the after-image table row to the corresponding after-image table row in the original ProDataSet temp-table. The AVM also sets the BEFORE-ROWID attribute of the row in the after-image table to the Unknown value (?), sets the ROW-STATE of the row in the after-image table to ROW-UNMODIFIED (0), and removes the before-image table row (if it has one).

Note: The after-image table row, that contains the changes to the corresponding after-image table row in the original ProDataSet temp-table, contains changes from the original ProDataSet temp-table as well as any changes made in the associated data source row based on the MERGE-BY-FIELD attribute and PREFER-DATASET attribute settings in effect during the save operation.

MESSAGE-AREA attribute
(Graphical interfaces only)

Controls the appearance of the message area in the window.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WINDOW widget

You can set this attribute only before the window is realized.

MESSAGE-AREA-FONT attribute

The font number of the font used in the message area of a window.

Data type: INTEGER

Access: Readable/Writeable

Applies to: WINDOW widget

The font number represents an entry in the font table maintained by the FONT-TABLE handle.

MIN-BUTTON attribute
(Windows only; Graphical interfaces only)

Determines whether the window has a minimize button in its caption bar.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WINDOW widget

In Windows, a window can have maximize and minimize buttons depending on the settings of the MIN-BUTTON and MAX-BUTTON attributes. Both buttons are created on the window. If you set the MIN-BUTTON to TRUE and the MAX-BUTTON to FALSE, only the MIN-BUTTON is enabled; the MAX-BUTTON is disabled.

The MIN-BUTTON attribute must be set before the window is realized. The default value is TRUE.

On character platforms, this attribute has no effect.

MIN-COLUMN-WIDTH-CHARS attribute
(Graphical interfaces only)

Sets the minimum width of a browse column in character units. If the browse has **not** been realized, all browse columns are minimally this size when realized. If the browse has been realized, any browse column smaller than the specified minimum is increased to the minimum width.

Data type: Decimal

Access: Readable/Writeable

Applies to: BROWSE widget

The default value for the MIN-COLUMNS-WIDTH-CHARS attribute is equivalent to 1 pixel and depends on the display resolution and the size of the default font of the system.

An error occurs if you attempt to programmatically set the width of a browse column to a value smaller than the value specified with the MIN-COLUMN-WIDTH-CHARS attribute.

If COLUMN-RESIZABLE is set to TRUE, the user cannot change a column's width to be less than the minimum width specified with the MIN-COLUMN-WIDTH-CHARS attribute.

The MIN-COLUMN-WIDTH CHARS attribute affects the FIT-LAST-COLUMN attribute. Therefore, if you set FIT-LAST-COLUMN to TRUE, the last browse column is only resized to fit within the viewport if its width is no smaller than the minimum width. See the FIT-LAST-COLUMN attribute for more information about the FIT-LAST-COLUMN attribute.

When you assign a decimal value to an attribute representing a measurement in character units, the AVM automatically rounds the assigned value to the nearest decimal value that corresponds to whole pixel units.

MIN-COLUMN-WIDTH-PIXELS attribute
(Graphical interfaces only)

Sets the minimum width of a browse column in pixels. If the browse has **not** been realized, all browse columns are minimally this size when realized. If the browse has been realized, any browse column smaller than the specified minimum is increased to the minimum width.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget

The default minimum browse column width is 1 pixel.

An error occurs if you attempt to programmatically set the width of a browse column to a value smaller than the value specified using the MIN-COLUMN-WIDTH-PIXELS attribute.

If COLUMN-RESIZABLE is set to TRUE, the user cannot change a column's width to be less than the minimum width specified with the MIN-COLUMN-WIDTH-PIXELS attribute.

The MIN-COLUMN-WIDTH PIXELS attribute affects the FIT-LAST-COLUMN attribute. As a result, if FIT-LAST-COLUMN is set to TRUE, the last browse column is only resized to fit within the viewport if its width is no smaller than the minimum width. See the FIT-LAST-COLUMN attribute for more information about the FIT-LAST-COLUMN.

MIN-HEIGHT-CHARS attribute

The minimum height of a window, in character units.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: BROWSE widget, WINDOW widget

MIN-HEIGHT-PIXELS attribute

The minimum height of a window, in pixels.

Data type: INTEGER

Access: Readable/Writeable

Applies to: WINDOW widget

MIN-SCHEMA-MARSHAL attribute

Set to TRUE to minimize schema information when marshaling data for a temp-table parameter. The temp-table may be an independent temp-table or a member of a ProDataSet object.

This attribute is supported only for backward compatibility. Use the SCHEMA-MARSHAL attribute instead.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Temp-table object handle

This attribute suppresses index descriptions and some field information (such as label, help, field validation expression, and so on) when marshaling data. It does marshal field names, data types, and extents.

The MIN-SCHEMA-MARSHAL attribute corresponds to the SCHEMA-MARSHAL attribute with a value of "MIN".

Note: If you specify both the MIN-SCHEMA-MARSHAL attribute and the SCHEMA-MARSHAL or NO-SCHEMA-MARSHAL attribute for an individual temp-table, the AVM uses the attribute you most recently specified.

Setting this attribute overrides the setting of the Temp-table Schema Marshal (`-ttmarshal`) startup parameter for an individual temp-table parameter. For more information about this startup parameter, see *Startup Command and Parameter Reference*.

See also

NO-SCHEMA-MARSHAL attribute, SCHEMA-MARSHAL attribute

MIN-VALUE attribute

The minimum value of a slider.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SLIDER widget

You can set this attribute only before the widget is realized.

MIN-WIDTH-CHARS attribute

The minimum width of a window, in character units.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: WINDOW widget

MIN-WIDTH-PIXELS attribute

The minimum width of a window, in pixels.

Data type: INTEGER

Access: Readable/Writeable

Applies to: WINDOW widget

MODIFIED attribute

Indicates whether the value of the SCREEN-VALUE attribute for the widget has changed.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), COMBO-BOX widget, EDITOR widget, FILL-IN widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

For browse columns, this attribute is readable only.

For all widgets, the MODIFIED attribute is set to TRUE when the SCREEN-VALUE attribute for the widget is changed, whether or not the field is enabled. For all widgets **except** the editor widget, the MODIFIED attribute is set to TRUE if the SCREEN-VALUE attribute for the widget is changed using an ABL statement, such as assignment or DISPLAY. DISPLAY sets MODIFIED to TRUE only when the field is enabled. You can then reset the attribute to FALSE for each widget that can receive input focus or otherwise change value after it is initially displayed.

For editors, the successful execution of either the SAVE-FILE( ) or the READ-FILE( ) methods sets the MODIFIED attribute to FALSE.

For browses, if any browse cell changes, the AVM sets MODIFIED to TRUE. The application can reset MODIFIED to FALSE as necessary. If the query associated with a browse is reopened, the AVM resets MODIFIED to FALSE.

If the widget is not already realized and you reference its MODIFIED attribute, the AVM realizes the widget.

You can use the SENSITIVE attribute to reset the MODIFIED attribute. For example, when the screen-value of a fill-in field is changed, MODIFIED is set to TRUE. If you then set SENSITIVE to FALSE and then to TRUE, you enable the fill-in for further input and the MODIFIED attribute is reset to FALSE.

MOUSE-POINTER attribute

Returns the name of the mouse pointer loaded by LOAD-MOUSE-POINTER( ).

Data type: CHARACTER

Access: Read-only

Applies to: BROWSE widget (browse and column), BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

MOVABLE attribute
(Graphical interfaces only)

Indicates whether the widget can receive direct manipulation events.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

Set MOVABLE to TRUE to enable users to move the widget. To enable users to move more than one widget at a time, you must also set the SELECTABLE attribute to TRUE for each widget.

Note: Setting the MOVABLE attribute to TRUE enables direct manipulation events for the widget. These events take precedence over all other events. This effectively prevents data entry using the widget until all direct manipulation events are disabled (that is, until MOVABLE, RESIZABLE, and SELECTABLE are all FALSE).

MOVE-AFTER-TAB-ITEM( ) method

Assigns the method widget to the tab position after a specified widget. Both the method widget and the specified widget must be in the same field group.

Return type: LOGICAL

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

Syntax

```
MOVE-AFTER[ -TAB-ITEM] ( handle )
```

handle

A handle to the widget after whose tab position you want to move the method widget.

If the operation is successful, the method returns TRUE. To set the first or last tab position, set the FIRST-TAB-ITEM or LAST-TAB-ITEM attribute (respectively) for the field group.

If *handle* specifies a frame, the tab order of the method widget is positioned so that it follows the last widget parented by the frame in that frames own tab order. For more information on how frames owned by a field group participate in the tab order of that field group, see the FRAME widget reference entry.

Note: Any tab reordering that you do with this method can be reset by a subsequent ENABLE statement unless you define the frame that owns the field group with the KEEP-TAB-ORDER option. For more information, see the ENABLE statement and Frame phrase reference entries.

MOVE-BEFORE-TAB-ITEM( ) method

Assigns the method widget to the tab position before a specified widget. Both the method widget and the specified widget must be in the same field group.

Return type: LOGICAL

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

Syntax

```
MOVE-BEFORE[ -TAB-ITEM] ( handle )
```

handle

A handle to the widget before whose tab position you want to move the method widget.

If the operation is successful, the method returns TRUE. To set the first or last tab position, set the FIRST-TAB-ITEM or LAST-TAB-ITEM attribute (respectively) for the field group.

If *handle* specifies a frame, the tab order of the method widget is positioned so that it precedes the first widget parented by the frame in that frames own tab order. For more information on how frames owned by a field group participate in the tab order of that field group, see the FRAME widget reference entry.

Note: Any tab reordering that you do with this method can be reset by a subsequent ENABLE statement unless you define the frame that owns the field group with the KEEP-TAB-ORDER option. For more information, see the ENABLE statement and Frame phrase reference entries.

MOVE-COLUMN( ) method
(Graphical interfaces only)

Repositions a column in a browse widget.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
MOVE-COLUMN ( source , destination )
```

source

An integer expression specifying the column to be moved.

destination

An integer expression specifying the position to which the column is moved.

The columns of a browse are numbered left to right beginning with 1 including both visible and hidden columns. For example, *browse*:MOVE-COLUMN(1, 3) moves the first column to the third position (the second column becomes the first column and the third column becomes the second column). If the column is successfully moved, the method returns the value TRUE.

MOVE-TO-BOTTOM( ) method

Moves the widget to the bottom (or back) of other widgets of the same class on the display.

Return type: LOGICAL

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

Syntax

```
MOVE-TO-BOTTOM ( )
```

For the purposes of this method, the classes are as follows:

- Windows
- Frames
- Images and rectangles
- All other field-level widgets

If the operation is successful, the method returns TRUE.

When you use this method, set the KEEP-FRAME-Z-ORDER attribute to TRUE.

In character interfaces, the MOVE-TO-BOTTOM method applies only to the Frame.

MOVE-TO-EOF( ) method

Moves the cursor position in an editor to the end of the current text.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
MOVE-TO-EOF ( )
```

If the operation is successful, the method returns TRUE.

MOVE-TO-TOP( ) method

Moves the widget to the top (or front) of other widgets of the same class on the display.

Return type: LOGICAL

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

Syntax

```
MOVE-TO-TOP ( )
```

For the purposes of this method, the classes are as follows:

- Windows
- Frames
- Images and rectangles
- All other field-level widgets

If the operation is successful, the method returns TRUE.

Images and rectangles are displayed behind other field-level widgets and cannot be moved on top of them.

When you use this method, set the KEEP-FRAME-Z-ORDER attribute to TRUE.

Note: In character interfaces, the MOVE-TO-TOP method applies only to the Frame.

MULTI-COMPILE attribute

Specifies whether ABL compiles all class definition files in the inherited class hierarchy or only those class definition files for which a cached version is not found.

Note: This attribute is applicable only when compiling class definition (`.cls`) files.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: COMPILER system handle

When set to TRUE, ABL compiles only those class definition files in the inherited class hierarchy that are not found in the cache. ABL also caches any classes or interfaces it compiles to avoid recompiling them during the session.

When set to FALSE, ABL compiles all class definition files in the inherited class hierarchy. ABL also clears the cache of any classes or interfaces compiled during the session. The default value is FALSE.

MULTIPLE attribute

Indicates the selection behavior of browse selection list widgets, and the read and write behavior of the system clipboard.

Data type: LOGICAL

Access: Readable/Writeable (Read-only for browse)

Applies to: BROWSE widget, CLIPBOARD system handle, SELECTION-LIST widget

For browse widgets, the MULTIPLE attribute specifies whether the user can select multiple rows from the widget, or only a single row. (Typically, the selected rows are processed in response to a DEFAULT-ACTION event.) The MULTIPLE attribute is read-only for browse widgets. The MULTIPLE attribute for a browse can be set before the browse is realized. Use the MULTIPLE or SINGLE option of the browse phrase in the DEFINE BROWSE statement to set the selection behavior for a browse widget. The MOUSE-SELECT-DOWN and MOUSE-SELECT-UP events are generated as the user scrolls through the browse.

Note: When an updateable browse is in edit mode, a cell has focus, all other selected rows are deselected.

For selection-list widgets, the MULTIPLE attribute specifies whether the user can select multiple items from the widget, or only a single item. (Typically, the selected rows are processed in response to a DEFAULT-ACTION event.) You can specify selection behavior for a selection list using the MULTIPLE or SINGLE option of a SELECTION-LIST phrase. You can set this attribute for a selection list only before the widget is realized.

Note: When a selection-list has the MULTIPLE attribute, the selection of an item does not clear any of the items previously selected. An item remains highlighted until the selection-list is cleared.

For the CLIPBOARD handle, the MULTIPLE attribute specifies whether the AVM reads and writes data to the CLIPBOARD handle as multiple items or as a single item. For more information, see the CLIPBOARD system handle reference entry.

MULTITASKING-INTERVAL attribute
(Windows only)

How often the AVM filters events between itself and other Windows applications.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SESSION system handle

The value of the MULTITASKING-INTERVAL attribute determines how often the AVM internally filters events (messages) between itself and other Windows applications. As the AVM filters these events more often, it executes procedures less efficiently, but allows other windows applications more opportunity to execute. Adjusting the internal event filter is particularly useful during background processing, such as report generation.

The default value, zero, tells the AVM never to filter events internally, giving ABL applications maximum access to execution resources. This is perfectly adequate for interactive ABL applications that block for input often, giving other applications enough opportunity to execute.

For values greater than zero, the lower the value, the more often the AVM internally filters events, giving other applications greater opportunity to execute, but slowing down the AVM execution. However, similar to a TRUE value for the IMMEDIATE-DISPLAY attribute, low non-zero values also cause the AVM to refresh the display more often, potentially providing crisper display interaction. Low non-zero values also provide better interoperability with other applications, for example, using Dynamic Data Exchange (DDE).

The maximum value you can set is 9999. In general, set this attribute greater than zero only for code segments that perform lengthy background operations, and reset it to zero before the application blocks for interactive input (for example, executes a WAIT-FOR or UPDATE statement). This attribute provides the same functionality as the MultitaskingInterval parameter in the current environment, which might be the Registry (Windows only) or an initialization file. For more information on environments, see the topics on user interface environments in *Manage ABL Applications*.

MUST-UNDERSTAND attribute

Indicates whether a SOAP-header-entryref object is mandatory (TRUE) or optional (FALSE) for the recipient to process.

Data type: LOGICAL

Access: Read-only

Applies to: SOAP-header-entryref object handle

If the SOAP-header-entryref object does not contain a MUST-UNDERSTAND attribute, this attribute returns FALSE.

Name property
(Windows only; Graphical interfaces only)

The name of the control-frame and control-frame COM object.

Return type: CHARACTER

Access: Readable/Writeable

Applies to: CONTROL-FRAME widget, COM object

Setting this value changes the NAME attribute of the corresponding control-frame widget to the same value.

Note: References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the Accessing COM object properties and methods.

Caution: If you change the value of this property at run time, any OCX event procedures that you have defined for a corresponding ActiveX control will not respond to control events because the events are sent with the new name.

NAME attribute

A string identifier for the specified object or widget.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Asynchronous request object handle, BROWSE widget (browse, column, and cell), Buffer object handle, Buffer-field object handle, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, Data-relation object handle, Data-source object handle, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, Procedure object handle, ProDataSet object handle, Query object handle, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, Server object handle, Server socket object handle, SLIDER widget, SOAP-header object handle, SOAP-header-entryref object handle, Socket object handle, SOURCE-PROCEDURE system handle, Stream object handle, SUB-MENU widget, TARGET-PROCEDURE system handle, Temp-table object handle, TEXT widget, THIS-PROCEDURE system handle, TOGGLE-BOX widget, WINDOW widget, X-document object handle, X-noderef object handle

For the SOAP-header and SOAP-header-entryref object handles, this attribute is read-only.

For static data representation widgets, the default value is the name of the field or variable associated with the widget. You can use the NAME attribute to store any information associated with the widget.

For a state-reset, state-aware, or stateless AppServer, this attribute returns the connection ID for the AppServer associated with the server handle. For a state-free AppServer, it returns the connection ID of the first AppServer connection created in the session pool with this server handle when the application service is first bound using the CONNECT( ) method. The default value is the unique connection name of the AppServer.

For Web services, the default value is the URL of the Web service procedure object from the WSDL. This is the portType name used on the RUN statement that instantiated this object.

For a non-Web service procedure, returns the pathname of the procedure file that contains the current procedure.

For control-frames, this attribute maps to the Name property of the of the control-frame COM object (ActiveX control container).

For dynamic widgets and asynchronous request handles, this attribute defaults to the Unknown value (?).

Caution: If you change the value of this property at run time, any OCX event procedures that you have defined for a corresponding ActiveX control will not respond to control events because the events are sent with the new name.

For query objects, the NAME attribute applies only to static queries. For more information on query objects, see *ABL Essentials.*.

For the SOAP-header object handle, this attribute is the qualified name of the SOAP-header object, which consists of a namespace prefix + ":" + HEADER.

For the SOAP-header-entryref object handle, this attribute is the qualified name of the SOAP-header-entryref object, which consists of a namespace prefix + ":" + localname. You cannot change this attribute directly; you must use the local-name and namespace-prefix.

The NAME attribute of the Temp-table object handle is writeable for dynamic and AVM-generated temp-tables. A new temp-table name cannot be assigned until after the TEMP-TABLE-PREPARE() method has been executed.

For the X-document object handle or X-noderef object handle, this attribute returns the name of the XML node.

For any object or widget, this attribute can contain any arbitrary value that you set.

NAMESPACE-PREFIX attribute

This attribute returns or sets the qualified part of a namespace-aware XML node name (that is, the prefix before the colon character). The prefix is used to identify elements that belong to the namespace associated with the prefix (as set by the NAMESPACE-URI attribute). For nodes created with the CREATE-NODE( ) method, or nodes of any type other than ELEMENT or ATTRIBUTE, this attribute returns the Unknown value (?).

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle, X-document object handle, X-noderef object handle

This attribute is read-only for the X-document object handle.

NAMESPACE-URI attribute

The namespace URI of a namespace-aware XML node name, a SOAP-header-entryref object, or a ProDataSet or Temp-Table element and its child elements. The namespace of an XML document is used to scope XML attributes and elements. For nodes created with the CREATE-NODE( ) method, or nodes of any type other than ELEMENT or ATTRIBUTE, this attribute returns the Unknown value (?).

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer object handle, ProDataSet object handle, SOAP-header-entryref object handle, Temp-table object handle, X-document object handle, X-noderef object handle

This attribute is read-only for the SOAP-header-entryref object handle, X-document object handle, and X-noderef object handle.

NEEDS-APPSERVER-PROMPT attribute

Indicates whether WebClient should prompt for AppServer connection parameters, if it does not find those values in the security cache, (TRUE) or not (FALSE).

Data type: LOGICAL

Access: Read-only

Applies to: CODEBASE-LOCATOR system handle

Valid only if LOCATOR-TYPE is "AppServer".

NEEDS-PROMPT attribute

Indicates whether WebClient should prompt for an Internet server userid and password, if it does not find those values in the security cache, (TRUE) or not (FALSE).

Data type: LOGICAL

Access: Read-only

Applies to: CODEBASE-LOCATOR system handle

NESTED attribute

Indicates whether child rows of a ProDataSet temp-table buffer are nested within their parent rows when writing the XML representation of a ProDataSet object that contains data-relations. This also causes the XML Schema definitions for the related temp-tables to be nested.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Data-relation object handle

See also

ADD-PARENT-ID-RELATION( ) method, DEFINE DATASET statement, WRITE-XML( ) method, WRITE-XMLSCHEMA( ) method

NEW attribute

Indicates whether the record in the buffer is newly created. If the record is newly created, NEW is TRUE. If the record in the buffer was read from the database, NEW is FALSE.

Data type: LOGICAL

Access: Read-only

Applies to: Buffer object handle

Note: The NEW attribute corresponds to the NEW function.

NEW-ROW attribute

Indicates whether the focused browse row exists in the database.

Data type: LOGICAL

Access: Read-only

Applies to: BROWSE widget

If this attribute is set to TRUE, the row in focus was added to the browse using the INSERT-ROW( ) method and has not been added to the database.

NEXT-COLUMN attribute

The handle of the next sibling, in physical order, of the current browse column whether or not the column is visible. The browse MOVE-COLUMN method changes the physical order of columns and updates this attribute accordingly.

Data type: HANDLE

Access: Read-only

Applies to: BROWSE widget (column)

NEXT-ROWID attribute

Provides the ROWID of the data source row at which the next FILL operation should start. The AVM sets this attribute after each FILL operation in a series of FILL operations to retrieve data source rows in batches. You typically assign the value of this attribute to the RESTART-ROWID attribute before each FILL operation.

Note: This attribute is not marshalled between the client and the application server. You are responsible for retrieving, storing, and transporting this attribute value between the client and the application server.

Data type: ROWID

Access: Readable/Writeable

Applies to: Data-source object handle

Syntax

```
NEXT-ROWID ( buffer-sequence-number | buffer-name )
```

buffer-sequence-number

An integer that represents the sequence number of a buffer in the list of buffers for the data-source object. Specify *buffer-sequence-number* to identify a buffer in the data-source object when the data-source object is defined against more than one database table buffer. The default is the first (or only) buffer in the data-source object.

Note: Sequence numbers for buffers in a data-source object start at one, where one represents the top level and subsequent numbers represent lower levels of join, if any.

buffer-name

A CHARACTER expression that evaluates to the name of a buffer in the list of buffers for the data-source object.

If an invalid buffer is specified, this attribute returns the Unknown value (?).

It is best to use the NEXT-ROWID attribute with a top-level ProDataSet temp-table, or a child temp-table that has only one parent record, because the AVM sets this attribute on the child temp-table for each parent record (as opposed to once per child temp-table).

Use this attribute when retrieving batches of data source rows containing stable data. Otherwise, it might be better to use a unique index to reopen the query associated with the data-source object to retrieve a specific batch of data source rows.

See also

BATCH-SIZE attribute, FILL( ) method, RESTART-ROWID attribute

NEXT-SIBLING attribute

The next entry in a list of handles, relative to a given handle.

Data type: HANDLE

Access: Read-only

Applies to: Asynchronous request object handle, BROWSE widget, Buffer object handle, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, Data-source object handle, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU-ITEM widget, Procedure object handle, ProDataSet object handle, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, Server object handle, SLIDER widget, Socket object handle, Server socket object handle, SESSION system handle, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

The following table summarizes the value of NEXT-SIBLING for each relevant handle type.

Table 91: NEXT-SIBLING attribute values by handle type

Handle type	Value of NEXT-SIBLING
Asynchronous Request	The handle of the next asynchronous request submitted for execution on the application server or Web Server that is running the specified request.
Procedure	The handle of the next persistent procedure in the current ABL session. If the current procedure is a proxy for a persistent procedure running on an application server or for a Web service, specifies the next procedure object bound to the same server handle.
Server	The next server handle created in the current ABL session (independent of subtype).
Socket and Server-socket	The next socket handle in the chain of socket handles for the current ABL session. Returns the Unknown value (?) for the last handle in the chain.
ProDataSet object	The handle to the next dynamic ProDataSet object in the chain of ProDataSet objects for the current ABL session, which is available after using the SESSION:FIRST-DATASET attribute.
Dynamic query	The handle to the next dynamic query in the chain of dynamic queries for the current ABL session, which is available after using the SESSION:FIRST-QUERY attribute.
Dynamic data-source object	The handle to the next dynamic data-source object in the chain of dynamic data-source objects for the current ABL session, which is available after using the SESSION:FIRST-DATA-SOURCE attribute.
Buffer object	The handle to the next buffer object in the session buffer list, which is available after using the SESSION:FIRST-BUFFER attribute.
Widget	The handle of the next widget in the widget list. **Note:** A widget must first be realized before it can become part of the list. A hidden widget cannot become part of the list since it is not realized. A widget that is already part of the list can be hidden and it remains part of the list.

If the given handle is the last handle in the list, NEXT-SIBLING assumes the value of an invalid handle. To check the validity of a handle, use the VALID-HANDLE function.

See also

PREV-SIBLING attribute

NEXT-TAB-ITEM attribute

The handle of the next widget in the tab order of a field group relative to the specified widget.

Data type: HANDLE

Access: Read-only

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

The NEXT-TAB-ITEM attribute returns the Unknown value (?) for a widget that is at the end of the tab order in a field group.

NO-CURRENT-VALUE attribute

The default behavior for a slider is to display the current value for a given position on a slider control. The NO-CURRENT-VALUE attribute allows you to override this default behavior.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SLIDER widget

NO-EMPTY-SPACE attribute
(Graphical interface only)

Allows the browse to display with no empty space to the right and no horizontal scroll bar.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

When the last browse column can be fully displayed in the browse viewport with empty space to the right, you can use NO-EMPTY-SPACE attribute to widen the width of the last browse column so that the column fits within the viewport with no empty space to its right and no horizontal scroll bar.

The default value is FALSE.

The following shows the DEFINE BROWSE statement syntax with NO-EMPTY-SPACE specified:

```
DEFINE BROWSE b1 QUERY q1
  DISPLAY Customer.CustNum Customer.Name
  ENABLE Customer.CustNum WITH 3 DOWN WIDTH 40 NO-EMPTY-SPACE
```

NO-EMPTY-SPACE is primarily intended for use in the initial layout of a static browse. It is most useful when laying out a browse with a specified width when you have only a few browse columns, and you want to fully use the available space in your viewport.

If the NO-EMPTY-SPACE is set to TRUE and there is empty space, the last browse column is widened to fill up the space. Also, if any browse column's width attribute is changed or the browse's width attribute is changed so that the last browse column is fully displayed in the browse's viewport with empty space to its right, then the last browse column's width is widened so that it fits within the viewport with no empty space and no horizontal scroll bar.

NO-EMPTY-SPACE never reduces the width of the last browse column.

NO-EMPTY-SPACE is ignored under the following circumstances:

- When the last browse column's width is explicitly set at run time after the browse is realized.
- When the last browse column displays partially or entirely outside of the viewport.
- If NO-EMPTY-SPACE is set to FALSE, the last browse column's width remains the same and is never changed by the AVM.
- If you specify NO-EMPTY-SPACE for an individual browse, and the `-expandbrow` startup parameter is also specified, then the NO-EMPTY-SPACE attribute overrides `-expandbrow` for that browse.
- When you use NO-EMPTY-SPACE, the original width of the last browse column is not remembered. For example, if the original width of the last browse column is 48 pixels, the DEFINE BROWSE statement has NO-EMPTY-SPACE specified, and if at run time there are 12 blank pixels as empty space in the right side of the viewport, the last browse column's width is increased to 60 pixels, so there is no blank space in the browse.
- If later at run time, the width of a column other than the last column is increased, a horizontal scroll bar is added to the browse. The width for the last browse column remains at 60 pixels.

NO-FOCUS attribute
(Windows only)

Determines whether a button can accept focus. A button for which the NO-FOCUS attribute is TRUE will not take focus when the mouse is clicked on it and it will not accept keyboard input. Also, the AVM will not generate ENTRY or LEAVE events for the button. NO-FOCUS buttons behave similarly to standard Windows toolbar buttons.

This attribute must be set before the button is realized. The default value is FALSE.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BUTTON widget

A button for which the NO-FOCUS attribute is TRUE will not be added to its parent frame's tab order. If the NO-FOCUS attribute is switched from TRUE to FALSE before the button is realized, the button will be added to the end of its parent frame's tab order. Switching the NO-FOCUS option from FALSE to TRUE before realization will remove the button from its parent frame's tab order.

The mnemonic key (**ALT** accelerator) for a widget will not work if the widget is removed from the tab order. Also, because the widget is not in the tab order, pressing **TAB** will not change focus from the widget.

Keep in mind that if a frame that contains a NO-FOCUS button does not itself have focus, the frame will not receive focus when the button is pushed. In this situation, frame entry or leave events are not generated. Focus stays on the current widget when a NO-FOCUS button is pushed, even across multiple frames in a window.

NONAMESPACE-SCHEMA-LOCATION attribute

Determines the location the XML Schema file to validate when elements do not contain a namespace.

Applies to: CHARACTER

Access: Readable/Writable

Applies to: X-document object handle, SAX-reader object handle

Contains the XML Schema file location for elements with no namespace.

This attribute specifies a single schema location. It defaults to an empty string ("").

NO-SCHEMA-MARSHAL attribute

Set to TRUE to exclude schema information when marshaling data for a temp-table parameter. The temp-table may be an independent temp-table or a member of a ProDataSet object.

This attribute is supported only for backward compatibility. Use the SCHEMA-MARSHAL attribute instead.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Temp-table object handle

The receiving procedure must have a temp-table definition in which to receive the temp-table. If the receiving temp-table is dynamic, and it does not have a definition, the AVM generates a run-time error.

Where this attribute suppresses index descriptions and all field information when marshaling data, the AVM cannot perform field validation. Be sure the schema of both the source and target temp-tables is the same.

The NO-SCHEMA-MARSHAL attribute corresponds to the SCHEMA-MARSHAL attribute with a value of "NONE".

Note: If you specify both the NO-SCHEMA-MARSHAL attribute and the SCHEMA-MARSHAL or MIN-SCHEMA-MARSHAL attribute for an individual temp-table, the AVM uses the attribute you most recently specified.

Setting this attribute overrides the setting of the Temp-table Schema Marshal (`-ttmarshal`) startup parameter for an individual temp-table parameter. For more information about this startup parameter, see *Startup Command and Parameter Reference*.

See also

MIN-SCHEMA-MARSHAL attribute, SCHEMA-MARSHAL attribute

NO-VALIDATE attribute

Specifies that the AVM ignore the validation conditions in the schema for all fields in a dynamic browse.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

This attribute is readable by both static and dynamic browses, and writeable only by dynamic browses.

If TRUE, the AVM does not run the validation for dynamic browse columns. If FALSE, the AVM runs the validation. The default value is FALSE.

Note: For a static browse, this attribute is set in the DEFINE BROWSE statement using the NO-VALIDATE option.

NODE-VALUE attribute

Returns (or sets) the value of the XML node.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: X-noderef object handle

The following example demonstrates the use of the NODE-VALUE attribute:

```
IF hNoderef:NODE-VALUE = "500" THEN
  hNoderef:NODE-VALUE = "1000".
```

NODE-VALUE-TO-LONGCHAR( ) method

Copies the contents of an XML X-noderef node to a LONGCHAR, and optionally converts the contents to a specific code page.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
NODE-VALUE-TO-LONGCHAR ( longchar [codepage ] )
```

longchar

An expression of type LONGCHAR.

codepage

A character-string expression that evaluates to the name of a code page. The name you specify must be a valid code page name available in the `DLC/convmap.cp` file. If you do not specify NODE-VALUE-TO-LONGCHAR( ) frees the memory currently allocated by *codepage* and the code page of *longchar* is fixed (that is, set using the FIX-CODEPAGE statement), the AVM converts *longchar* to the fixed code page. If you do not specify *codepage* and the code page of *longchar* is not fixed, the AVM converts *longchar* to the code page specified by `-cpinternal`. If you specify

codepage and the code page of *longchar* is fixed, they must agree. Otherwise, the AVM raises a run-time error.

longchar (if any), allocates sufficient memory to the LONGCHAR to accommodate the node, and copies the node to the LONGCHAR.

If X-NODEREF:NODE-VALUE is the empty string (""), the resulting *longchar*NODE-VALUE-TO-LONGCHAR( ) has a size of zero length.

For more information on accessing XML documents using the Document Object Model (DOM) interface, see *Use XML with ABL Applications*.

NODE-VALUE-TO-MEMPTR( ) method

Copies the contents of an XML X-noderef node to a MEMPTR. This makes it easier to manipulate when its length exceeds the ABL limit for text strings, which is approximately 32K.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
NODE-VALUE-TO-MEMPTR ( memptr )
```

memptr

An expression of type MEMPTR.

NODE-VALUE-TO-MEMPTR( ) frees the memory currently allocated by *memptr* (if any), allocates sufficient memory to the MEMPTR to accommodate the node, and copies the node to the MEMPTR.

If X-NODEREF:NODE-VALUE is "" (the empty string), the resulting *memptr* has a size of zero.

Note: When you no longer need the memory used by *memptr*, you must free it yourself. To do so, use the SET-SIZE statement.

The following fragment uses NODE-VALUE-TO-MEMPTR( ) to access a large text node in chunks:

```
IF my-xnoderef:SUBTYPE = "#TEXT" THEN DO:
  IF LENGTH(my-xnoderef:NODE-VALUE, "RAW") > 32000 THEN DO:
    ASSIGN
      succes = my-xnoderef:NODE-VALUE-TO-MEMPTR(my-memptr)
      pos    = 1
      len    = 32000.
    DO WHILE pos < GET-SIZE(my-memptr):
      my-nodeval = GET-STRING(my-memptr, pos, len).
      /* Do something with my-nodeval */
      pos = pos + len.
    END.
    SET-SIZE(my-memptr) = 0.
  END.
  ELSE
    my-nodeval = my-xnoderef:NODE-VALUE.
END.
```

For more information on accessing XML documents using the Document Object Model (DOM) interface, see *Use XML with ABL Applications*.

NORMALIZE( ) method

Normalizes TEXT and ATTRIBUTE nodes in the full depth of the sub-tree under this XML node.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
NORMALIZE ( )
```

The NORMALIZE( ) method normalizes TEXT nodes by removing empty TEXT nodes and merging adjacent TEXT nodes. Thus, only structure node types (such as ELEMENT, CDATA-SECTION, and so on) separate TEXT nodes. The NORMALIZE( ) method also normalizes white space in ATTRIBUTE nodes according to the rules defined by the XML specification.

NUM-BUFFERS attribute

The number of buffers in a query or ProDataSet object.

Data type: INTEGER

Access: Read-only

Applies to: ProDataSet object handle, Query object handle

NUM-BUTTONS attribute

The number of items in a radio set.

Data type: INTEGER

Access: Read-only

Applies to: RADIO-SET widget

NUM-CHILD-RELATIONS attribute

The number of **active** relations for which the buffer is the parent. A buffer may be a parent in multiple relations, but a child in only one.

Data type: INTEGER

Access: Read-only

Applies to: Buffer object handle

NUM-CHILDREN attribute

Returns the number of child nodes below the node referred to by a node reference. Attributes are not counted since they are not considered children of a node.

Data type: INTEGER

Access: Read-only

Applies to: X-document object handle, X-noderef object handle

The following example demonstrates getting all the child nodes from the XML node referenced by hNoderef using the NUM-CHILDREN attribute:

```
REPEAT jx = 1 TO hNoderef:NUM-CHILDREN:
  hNoderef:GET-CHILD(hNoderefChild, jx).
  . . .
END.
```

NUM-COLUMNS attribute

The number of columns in a browse. This number includes hidden as well as visible columns.

Data type: INTEGER

Access: Read-only

Applies to: BROWSE widget

NUM-DROPPED-FILES attribute
(Windows only; Graphical interfaces only)

Indicates the number of files dropped in the last drag-and-drop operation performed on the widget.

Data type: INTEGER

Access: Read-only

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

If there is no current drag-and-drop operation, NUM-DROPPED-FILES returns the Unknown value (?).

NUM-ENTRIES attribute
(Graphical interfaces only)

The number of entries in a color table or font table.

Data type: INTEGER

Access: Readable/Writeable

Applies to: COLOR-TABLE system handle, FONT-TABLE system handle

This attribute returns zero (0) in character interfaces because colors and fonts are not supported for character interfaces.

NUM-FIELDS attribute

The number of fields defined in the buffer's table.

Data type: INTEGER

Access: Read-only

Applies to: Buffer object handle

NUM-FORMATS attribute

The number of formats available for reading the data currently stored in the clipboard.

Data type: INTEGER

Access: Read-only

Applies to: CLIPBOARD system handle

If there are no formats available, the attribute returns 0. For more information, see the reference entry for the CLIPBOARD system handle.

NUM-HEADER-ENTRIES attribute

The number of SOAP-header-entryref object entries attached to the SOAP-header object.

Data type: INTEGER

Access: Read-only

Applies to: SOAP-header object handle

NUM-ITEMS attribute

The number of entries in a combo box, SAX-attributes object, or selection list.

Data type: INTEGER

Access: Read-only

Applies to: BROWSE widget (column), COMBO-BOX widget, SAX-attributes object handle, SELECTION-LIST widget

For browses, this attribute applies only to combo-box browse columns.

NUM-ITERATIONS attribute (data objects)

Indicates how many levels deep you are in a recursive FILL of a ProDataSet.

Data type: INTEGER

Access: Read-only

Applies to: Buffer object handle

As a recursive ProDataSet FILL is proceeding, it creates clones of the relevant buffers, relations, queries and data-sources for each level of recursion. As a new record is added to the ProDataSet, it fires FILL events on the recursed, cloned buffer. From inside the event handler, you may want to see previous iterations of the buffer, its parent, grandparent, great-grandparent, and so on. NUM-ITERATIONS indicates the level of the recursion.

If the buffer object handle does not point to a ProDataSet temp-table or is not part of a recursive data-relation, NUM-ITERATIONS returns the Unknown value (?).

See also

ADD-PARENT-ID-RELATION( ) method, CURRENT-ITERATION attribute (Data Objects), DEFINE DATASET statement, GET-ITERATION( ) method (Data Objects), RECURSIVE attribute

NUM-ITERATIONS attribute (widget objects)

The number of currently visible foreground iterations for a frame or the number of rows currently visible in a browse widget.

Data type: INTEGER

Access: Read-only

Applies to: BROWSE widget, FRAME widget

NUM-LINES attribute

The number of lines in an editor widget.

Data type: INTEGER

Access: Read-only

Applies to: EDITOR widget

Lines are substring of the editor field or variable that are terminated by end-of-line characters. The editor inserts end-of-line characters at the current cursor position it receives a RETURN event.

NUM-LOCKED-COLUMNS attribute

The number of visible leading columns locked in a browse widget. If a locked column is hidden, the next visible non-locked column in the browse will then become locked.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget

When you use the horizontal scrollbar to scroll columns in the browse, locked columns do not move. For example, if NUM-LOCKED-COLUMNS is 3, then the three leftmost columns in the browse are locked.

Note: In character mode, this attribute can only be set before the widget is realized.

NUM-LOG-FILES attribute

The number of rolled over log files to keep on disk at any one time, for ABL session, including the current log file.

Data type: INTEGER

Access: Read-only

Applies to: DSLOG-MANAGER system handle, LOG-MANAGER system handle

Valid values are:

- **0** — This means there is no limit on the number of log files to keep.
- **2 or greater** — The default is 3.

The `NUM-LOG-FILES` attribute corresponds to the Number of Log Files to Keep (`-numlogfiles`) startup parameter.

For ABL clients, use the LOG-THRESHOLD attribute of the `LOG-MANAGER` or the Log Threshold (`-logthreshold`) startup parameter to specify the file size at which OpenEdge rolls over (renames and saves) log files.

For DataServers, use the Log Threshold (`-logthreshold`) startup parameter to specify the file size at which the DataServer context will roll over (renames and saves) its log files when using self-service clients or ProBroker.

For more information about the Number of Log Files to Keep (`-numlogfiles`) and Log Threshold (`-logthreshold`) startup parameters, see *Startup Command and Parameter Reference*, *Use the Microsoft SQL Data Server*, and *Troubleshoot ABL Applications*.

NUM-MESSAGES attribute

The number of error messages currently available through the specified system handle.

Data type: INTEGER

Access: Read-only

Applies to: COMPILER system handle, ERROR-STATUS system handle

For the COMPILER system handle, this value represents the number of compilation errors detected during the preceding ABL source code compilation.

For the ERROR-STATUS system handle, this value represents the number of messages returned for the ERROR condition raised in a statement run with the NO-ERROR option.

NUM-PARAMETERS attribute

The number of parameters expected.

Data type: INTEGER

Access: Readable/Writable

Applies to: Call object handle

Syntax

```
NUM-PARAMETERS ( integer-expression )
```

integer-expression

An integer expression indicating the number of parameters expected. The default is zero.

Whenever NUM-PARAMETERS is set, all existing parameters, including those from earlier uses of SET-PARAMETER, are cleared and deallocated.

If there are parameters to be passed, NUM-PARAMETERS must be set before the INVOKE( ) method is executed.

When you are getting a parameter, set NUM-PARAMETERS to the actual number of parameters to be passed, which might vary if one or more trailing parameters are optional.

When you are setting an attribute, use NUM-PARAMETERS 1.

NUM-REFERENCES attribute

The number of references to a buffer, ProDataSet, or temp-table object that is defined as a parameter to which reference-only objects are bound.

Data type: INTEGER

Access: Read-only

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

Use this attribute to determine whether a buffer, ProDataSet, or temp-table object is referenced by another procedure before you delete the defining procedure or the referenced object itself (if it is a dynamic object).

If the buffer, ProDataSet, or temp-table object is not referenced by any other procedure (or other such object), this attribute returns 0. Otherwise, this attribute returns the number of procedures (or other such objects) currently referencing the object.

This attribute applies to objects defined as reference-only parameters, not shared objects.

See also

DEFINE DATASET statement (REFERENCE-ONLY option), DEFINE TEMP-TABLE statement (REFERENCE-ONLY option), DELETE OBJECT statement, DELETE PROCEDURE statement, RUN statement

NUM-RELATIONS attribute

The number of data-relation objects in a ProDataSet object.

Data type: INTEGER

Access: Read-only

Applies to: ProDataSet object handle

NUM-REPLACED attribute

Indicates the number of occurrences replaced by the last REPLACE( ) method executed for the Editor.

Data type: INTEGER

Access: Read-only

Applies to: EDITOR widget

If the Editor has not yet been realized, the attribute has the Unknown value (?).

NUM-RESULTS attribute

The number of rows currently in a query's result list.

Data type: INTEGER

Access: Read-only

Applies to: Query object handle

Note: The NUM-RESULTS attribute corresponds to the NUM-RESULTS function.

See also

NUM-RESULTS function

NUM-SELECTED-ROWS attribute

The number of rows currently selected in a browse widget.

Data type: INTEGER

Access: Read-only

Applies to: BROWSE widget

A browse can have more than one row selected only if the MULTIPLE attribute is TRUE.

NUM-SELECTED-WIDGETS attribute

The number of top-level widgets in a frame or window that the user has selected for direct manipulation.

Data type: INTEGER

Access: Read-only

Applies to: DIALOG-BOX widget, FRAME widget, WINDOW widget

For a window, this attribute returns the number of selected frames. For a frame or dialog box, this attribute returns the number of selected field-level widgets. You can use the GET-SELECTED-WIDGET( ) method to access the individual selected widgets.

NUM-SOURCE-BUFFERS attribute

The number of source buffers in the data-source object.

Data type: INTEGER

Access: Read-only

Applies to: Data-source object handle

NUM-TABS attribute

The number of widgets in the field group with tab positions.

Data type: INTEGER

Access: Read-only

Applies to: FIELD-GROUP widget

NUM-TO-RETAIN attribute

The number of frame iterations to retain when a down frame scrolls to a new set of iterations.

Data type: INTEGER

Access: Read-only

Applies to: FRAME widget

This value is set using the RETAIN option of the Frame phrase.

NUM-TOP-BUFFERS attribute

The number of top-level buffers in a ProDataSet object.

Note: A top-level buffer is a ProDataSet object buffer that is not a child in any active data relation. There may be one or more top-level buffers in a ProDataSet object.

Data type: INTEGER

Access: Read-only

Applies to: ProDataSet object handle

NUM-VISIBLE-COLUMNS attribute

Returns the number of visible columns in a browse.

Data type: INTEGER

Access: Read-only

Applies to: BROWSE widget

NUMERIC-DECIMAL-POINT attribute

The character that represents, in formatted text, a number's decimal point.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

NUMERIC-FORMAT attribute

How to interpret commas and periods within numeric values.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SESSION system handle

The possible values are "`American`", "`European`" or a character string consisting of the thousands separator followed by the decimal point. This attribute provides the same functionality as the European Numeric Format (`-E`) parameter.

Note: Although NUMERIC-FORMAT remains writable, it accepts only the values "`European`" and "`American`." To change the thousands separator or the decimal point in formatted text, use the new SET-NUMERIC-FORMAT( ) method of the SESSION system handle.

NUMERIC-SEPARATOR attribute

The character that represents, in formatted text, a number's thousands separator.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

ON-FRAME-BORDER attribute

Indicates whether the last event was a mouse event that occurred on a frame border.

Data type: LOGICAL

Access: Read-only

Applies to: LAST-EVENT system handle

OPTIONS attribute

Indicates the options enabled that control the behavior of the ABL compiler. The default value is the empty string ("") or the options specified by the `-compileroptionsfile` startup parameter. See OPTIONS under the COMPILE statement entry for a complete list of possible values.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: COMPILER system handle

The options are set and stored as a comma-separated list. For example, this code sets the options that require full field names (generating warnings but allowing compilation after failures) and full keywords (generating errors and preventing compilation when there are failures).

```
COMPILER:OPTIONS = "require-full-names,require-full-keywords:Error".
```

The value of the OPTIONS attribute is overridden when OPTIONS or OPTIONS-FILE is specified for a COMPILE statement. If one or more options in the attribute are invalid, the attribute keeps the value it had before the attempted assignment.

ORDINAL attribute

Specifies the number of the entry point (the nth routine) of the Windows Dynamic Link Library (DLL) to invoke.

Data type: INTEGER

Access: Readable/Writable

Applies to: Call object handle

Using the call object handle, there are two alternate methods for invoking a DLL routine—either by specifying the routine's name with the CALL-NAME attribute or by specifying the routine's position in the DLL with the ORDINAL attribute. You can specify an ORDINAL value or a CALL-NAME value, but not both. Specifying an ORDINAL value and setting CALL-NAME at the same time causes the AVM to raise error at run time.

The AVM raises error at run time if the ORDINAL entry point does not match any routine in a given DLL.

For UNIX shared library routines, this option does not apply and is ignored.

See also

CALL-NAME attribute

ORIGIN-HANDLE attribute

Returns the handle of the temp-table in the original source ProDataSet object that corresponds to the temp-table currently associated with this temp-table handle.

Data type: HANDLE

Access: Read-only

Applies to: Temp-table object handle

The AVM uses this value to match up temp-tables in a MERGE-CHANGES operation.

Note: If you invoke the EMPTY-DATASET( ) or EMPTY-TEMP-TABLE( ) method on the original source ProDataSet or its corresponding temp-table, respectively, the ABL Virtual Machine (AVM) sets this attribute to the Unknown value (?).

ORIGIN-ROWID attribute

Returns the ROWID of the row in the original before-image table that corresponds to the row in the change table currently associated with this buffer handle.

Data type: ROWID

Access: Read-only

Applies to: Buffer object handle

The AVM uses this value to match up temp-table rows in a MERGE-CHANGES operation.

OVERLAY attribute

Indicates whether the frame can overlay other frames on the display.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget

If the OVERLAY attribute is TRUE, the frame can overlay any other frame that does not have its TOP-ONLY attribute set to TRUE.

OWNER attribute

The handle of the widget that owns a menu widget.

Data type: HANDLE

Access: Read-only

Applies to: MENU widget

For a menu bar, the OWNER attribute returns the window with which the menu bar is associated. For a pop-up menu, the OWNER attribute returns the widget with which the menu is associated.

OWNER-DOCUMENT attribute

Returns the handle of the owning document of a node.

Data type: HANDLE

Access: Read-only

Applies to: X-noderef object handle

The following example demonstrates the use of the OWNER-DOCUMENT attribute:

```
DEFINE VARIABLE hDoc  AS HANDLE NO-UNDO.
DEFINE VARIABLE hDoc2 AS HANDLE NO-UNDO.

hDoc:LOAD("file", "my.xml", TRUE).
hDoc:GET-DOCUMENT-ELEMENT(hNoderef).
hDoc2 = hNoderef:OWNER-DOCUMENT.

/* At this point, hDoc2 and hDoc should be the same. */
```

PAGE-BOTTOM attribute

Indicates whether a frame is a footer frame in paged output.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget

If PAGE-BOTTOM is TRUE, the frame appears at the end of each page of output.

PAGE-TOP attribute

Indicates whether a frame is a header frame in paged output.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget

If PAGE-TOP is TRUE, the frame appears at the beginning of each page of output.

PARAMETER attribute

The value of the Parameter (`-param`) startup parameter specified for the current session.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

Use the Parameter (`-param`) parameter to specify a character string that can be accessed from ABL procedures. The AVM does not check the value of the PARAMETER attribute and you can use the parameter and attribute to store any arbitrary string value.

PARENT attribute

The handle of the parent of a widget.

Data type: HANDLE

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, SHADOW-WINDOW widget, WINDOW widget

This attribute is read-only for field groups.

For field-level widgets, the parent widget is the field group that contains the widget. For field groups, the parent widget is the frame that contains the field group. For frames, the parent widget is the window or field group that contains the frame. For a submenu or menu item, the parent widget is the menu or submenu that contains the submenu or menu item.

For ABL windows, the parent widget is another window that parents this window. The main feature of these window families is that when the parent window is minimized, all of its child windows are hidden. .NET forms have a similar feature for parenting forms to other forms using the `Owner` property or the `AddOwnedForm( )` method, which in OpenEdge are both provided by the Progress.Windows.Form class.

However, .NET form families exhibit one feature that ABL window families do not share—child forms never appear behind the parent form. The reason is that ABL window family behavior results from a native OpenEdge implementation that is different from the feature provided by Windows. In an ABL session, the .NET form and ABL window family mechanisms continue to work differently for forms and windows as long as .NET forms only parent forms and ABL windows only parent windows.

However, you can also parent .NET forms and ABL windows to each other using the PARENT attribute on the shadow window of a .NET form or the PARENT attribute on an ABL window. You can obtain the handle to the shadow window of a .NET form that you want to be a child or a parent from the value of the ProWinHandle property on the form (provided by Progress.Windows.Form). Then:

- To parent the form as a child of a window, set the form shadow window's PARENT attribute to the handle of the ABL window you want to become the form's parent.

- To parent a child window to this form, set the PARENT attribute of the ABL window you want to become the form's child to the shadow window handle of this parent form.

When you establish parent/child relationships like this between .NET forms and ABL windows, both the forms and windows in these mixed form and window families conform to the ABL window family mechanism (which allows the children to appear behind the parent) instead of conforming to the .NET form family mechanism (which forces the children to appear in front of the parent).

Note: You cannot use the PARENT attribute of form shadow windows to parent .NET forms to other .NET forms in an ABL session. You can only establish standard .NET parent/child relationships among forms using the `Owner` property or `AddOwnedForm( )` method of .NET forms.

See also

Progress.Windows.Form class, ProWinHandle property

PARENT-BUFFER attribute

Returns the buffer handle of the parent member of the data-relation object.

Data type: HANDLE

Access: Read-only

Applies to: Data-relation object handle

PARENT-FIELDS-AFTER attribute

Returns a comma-separated list of fields from the parent record in the relation to be written to XML or JSON after all nested child records. The attribute returns the Unknown value (?) if it is not specified for the data-relation object.

Data type: CHARACTER

Access: Read-only

Applies to: Data-relation object handle

See also

PARENT-FIELDS-BEFORE attribute

PARENT-FIELDS-BEFORE attribute

Returns a comma-separated list of fields from the parent record in the relation to be written to XML or JSON before any nested child records. The attribute returns the Unknown value (?) if the data-relation object is not specified.

Data type: CHARACTER

Access: Read-only

Applies to: Data-relation object handle

See also

PARENT-RELATION attribute

PARENT-ID-RELATION attribute

Returns a TRUE or FALSE value that indicates whether the relationship between parent and child records is based on the RECID of the parent record. If PARENT-ID-RELATION is TRUE, the value of RELATION-FIELDS attribute will be the name of the child buffer's RECID field.

Data type: LOGICAL

Access: Read-only

Applies to: Data-relation object handle

See also

RELATION-FIELDS attribute

PARENT-RELATION attribute

Returns the handle to the SELECTION data-relation object for the parent of this buffer. Since a buffer may be a child in only one relation, there can be only one parent for any buffer.

Data type: HANDLE

Access: Read-only

Applies to: Buffer object handle

PARSE-STATUS attribute

The current status of a SAX parse.

Data type: INTEGER

Access: Read-only

Applies to: SAX-reader object handle

The default value is SAX-UNINITIALIZED.

The values that PARSE-STATUS can assume are described in the following table.

Table 92: PARSE-STATUS attribute values

This value . . .	Indicates . . .
SAX-UNINITIALIZED	No parsing has occurred.
SAX-RUNNING	Parsing has begun.

This value . . .	Indicates . . .
SAX-COMPLETE	Parsing has begun and one of the following has occurred: • The parser has determined that there are no more tokens in the XML source. • The application has stopped the parser by calling the STOP-PARSING( ) method.
SAX-PARSER-ERROR	One of the following has occurred: • The parser could not start or could not continue. Perhaps the parser could not be loaded, the XML source could not be found, the XML source was invalid, etc. • The parser started, but a callback executed a RETURN ERROR statement.

PASSWORD-FIELD attribute

Displays password data in a field as a series of fill characters.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FILL-IN widget

If TRUE, the current value of a fill-in field or any character value typed into the fill-in field is displayed as a series of fill characters. The default value is FALSE.

In Windows GUI platforms, the default fill character is the asterisk (*). On non-Windows GUI or character platforms, the default fill character is a blank.

PATHNAME attribute

The absolute or relative pathname of the file specified by the FILE-NAME attribute of the FILE-INFO Handle.

Data type: CHARACTER

Access: Read-only

Applies to: FILE-INFO system handle

If the FILE-NAME attribute specifies a simple filename or relative pathname, this attribute returns a relative pathname based on the PROPATH. Otherwise, it returns the absolute pathname specified in FILE-NAME. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

PBE-HASH-ALGORITHM attribute

A text string containing the name of the hash algorithm to use with the GENERATE-PBE-KEY function to generate a password-based encryption key.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SECURITY-POLICY system handle

The method accepts the following values and corresponding hash algorithms in increasing order of security and decreasing order of performance:

- `"MD5"` — RSA Message Digest Hash Algorithm (MD5). Provided for backward compatibility only and should not be used in new development.
- `"SHA-1"` — United States Government Secure Hash Algorithm (SHA-1).
- `"SHA-256"` — United States Government Secure Hash Algorithm (SHA-256).
- `"SHA-512"` — United States Government Secure Hash Algorithm (SHA-512).

The default value is `"SHA-1"`.

You are responsible for generating, storing, and transporting this value.

See also

MESSAGE-DIGEST function

PBE-KEY-ROUNDS attribute

The number of hash algorithm iterations to perform in the GENERATE-PBE-KEY function to generate a password-based encryption key. The value must be a positive integer. The default value is 1000.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SECURITY-POLICY system handle

You are responsible for generating, storing, and transporting this value.

Note: Setting the PBE-KEY-ROUNDS attribute to an extremely large number can significantly slow the performance of the GENERATE-PBE-KEY function.

PERSISTENT attribute

Indicates whether the procedure is persistent.

Data type: LOGICAL

Access: Readable/Writable

Applies to: Call object handle, Procedure object handle, SOURCE-PROCEDURE system handle, TARGET-PROCEDURE system handle, THIS-PROCEDURE system handle

The PERSISTENT attribute is TRUE when the RUN statement that executes a procedure is invoked with the PERSISTENT option. Otherwise, it is FALSE.

When a dynamic invoke returns and PERSISTENT is TRUE, the IN-HANDLE attribute contains a handle to the running persistent procedure.

When a Windows DLL or UNIX shared library is invoked dynamically and PERSISTENT is TRUE, the DLL or shared library remains loaded in memory until the AVM exits or the session executes the RELEASE EXTERNAL statement

Returns TRUE for a Web service procedure.

If a procedure has already been invoked with PROCEDURE-TYPE set to "PERSISTENT", then PERSISTENT will automatically be set to TRUE. Setting PERSISTENT to TRUE when PROCEDURE-TYPE has already been set as "SINGLE-RUN" or "SINGLETON" will result in a run-time error.

See also

IN-HANDLE attribute, INVOKE( ) method (Handle), PROCEDURE-TYPE attribute, RELEASE EXTERNAL statement, RUN statement

PERSISTENT-CACHE-DISABLED attribute

Indicates whether WebClient disables the saving of security cache attribute values between sessions (TRUE) or not (FALSE).

Data type: LOGICAL

Access: Read-only

Applies to: CODEBASE-LOCATOR system handle

When TRUE, KEEP-SECURITY-CACHE will be FALSE.

PERSISTENT-PROCEDURE attribute

For the application server, this attribute returns the proxy remote persistent procedure handle of the remote procedure that contains the internal procedure executed for the specified asynchronous request. For Web services, this attribute returns the Web service procedure object handle.

Data type: HANDLE

Access: Read-only

Applies to: Asynchronous request object handle

This handle is the same as the handle specified by the IN *proc-handle* option of the RUN statement that executes this request. If the request is running a remote external (not internal) procedure, this attribute contains an invalid handle.

PFCOLOR attribute
(Character interfaces only)

The color number of the color of a widget that has input focus. The edge color of a rectangle widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse, column, and cell), BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, MENU widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TOGGLE-BOX widget, WINDOW widget

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

For field-level widgets that receive focus, the PFCOLOR attribute specifies the input color for the widget. In windows, the PFCOLOR attribute specifies the color inherited by menu items in the menu bar when they are chosen, if the menu items don't already have the PFCOLOR specified.

For browse widgets, this color represents the input color for the focused cell.

For more information on widget color, see the DCOLOR attribute.

PIXELS-PER-COLUMN attribute

The number of pixels in each column of the display.

Data type: INTEGER

Access: Read-only

Applies to: SESSION system handle

This value is also the pixel size of a horizontal character unit, and depends on the resolution of the display and the size of the default system font.

PIXELS-PER-ROW attribute

The number of pixels in each row of the display.

Data type: INTEGER

Access: Read-only

Applies to: SESSION system handle

This value is also the pixel size of a vertical character unit, and depends on the resolution of the display and the size of the default system font.

POPUP-MENU attribute

The pop-up menu associated with a widget.

Data type: HANDLE

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

The value you assign to POPUP-MENU must be the handle of a previously defined menu whose POPUP-ONLY attribute is TRUE.

POPUP-ONLY attribute

Indicates whether a menu is pop-up or a menu bar.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: MENU widget

Set the POPUP-ONLY attribute to TRUE to use the menu as a pop-up menu. Otherwise, the menu is a menu bar that you can associate with a window. FALSE is the default value. You can set this attribute only before the menu is realized.

POSITION attribute

The position of a buffer-field within the database record.

Data type: INTEGER

Access: Read-only

Applies to: Buffer-field object handle

Note: The POSITION attribute applies to OpenEdge databases only.

PREFER-DATASET attribute

Specifies whether the AVM ignores modifications to the data currently in the data source when saving changes from a ProDataSet temp-table buffer to the associated data source using the SAVE-ROW-CHANGES( ) method. The default value is FALSE.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Data-source object handle

If TRUE, the AVM ignores the data currently in the data source and automatically accepts the data from the ProDataSet temp-table buffer.

If FALSE, the AVM compares the before-image of the data in the ProDataSet temp-table buffer, saved while tracking changes for the buffer, to the corresponding data source buffer to determine whether the data in the data source has changed since being read. The AVM evaluates any data source changes to determine whether or not a data conflict exists based on the MERGE-BY-FIELD attribute setting (that is, on either a field-to-field or buffer-to-buffer basis).

PREPARED attribute

This attribute returns TRUE if the TEMP-TABLE-PREPARE( ) method has been called with no subsequent CLEAR( ) method. That is, it is true when the temp-table is in the PREPARED state.

Data type: LOGICAL

Access: Read-only

Applies to: Temp-table object handle

PREPARE-STRING attribute

The character string passed to the most recent QUERY-PREPARE. If QUERY-PREPARE was not called, or the query was just opened with the OPEN QUERY statement, PREPARE-STRING has the Unknown value (?).

Data type: CHARACTER

Access: Read-only

Applies to: Query object handle

For an example, see the reference entry for the INDEX-INFORMATION attribute.

PREV-COLUMN attribute

The handle of the previous sibling, in physical order, of the current browse column whether or not the column is visible. The browse MOVE-COLUMN method changes the physical order of columns and updates this attribute accordingly.

Data type: HANDLE

Access: Read-only

Applies to: BROWSE widget (column)

PREV-SIBLING attribute

The previous entry in the list of handles, relative to a given handle.

Note: Returns the Unknown value (?) for a Web service procedure.

Data type: HANDLE

Access: Read-only

Applies to: Asynchronous request object handle, BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU-ITEM widget, Procedure object handle, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, Server object handle, SLIDER widget, Socket object handle, Server socket object handle, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

The following table summarizes the value of PREV-SIBLING for each relevant handle type.

Table 93: PREV-SIBLING attribute values by handle type

Handle type	Value of PREV-SIBLING
Asynchronous Request	The handle of the previous asynchronous request submitted for execution on the application server or Web Server that is running the specified request.
Procedure	The handle of the previous persistent procedure in the current ABL session. If the current procedure is a proxy for a persistent procedure running on an application server or for a Web service, specifies the previous procedure object bound to the same server handle.
Server	The previous server handle created in the current ABL session (independent of subtype).
Socket and Server-socket	The previous socket handle in the chain of socket handles for the current ABL session. Returns the Unknown value (?) for the first handle in the chain.
Widget	The handle of the previous widget in the widget list. **Note:** A widget must first be realized before it can become part of the list. A hidden widget cannot become part of the list since it is not realized. A widget that is already part of the list can be hidden and it remains part of the list.

If the given handle is the first handle in the list, PREV-SIBLING assumes the value of an invalid handle. To check the validity of a handle, use the VALID-HANDLE function.

See also

NEXT-SIBLING attribute

PREV-TAB-ITEM attribute

The handle of the previous widget in the tab order of a field group relative to the specified widget.

Data type: HANDLE

Access: Read-only

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

The PREV-TAB-ITEM attribute returns the Unknown value (?) for a widget that is at the beginning of the tab order in a field group.

PRIMARY attribute

This attribute sets or returns the name of the primary index of the temp-table or buffer.

Data type: CHARACTER

Access: Readable/Writable

Applies to: Buffer object handle, Temp-table object handle

Notes

- When used with a temp-table object handle, PRIMARY can only be updated before the TEMP-TABLE-PREPARE( ) method has been called. It returns the Unknown value (?) if the temp-table is not in a PREPARED state.
- When used with a buffer object handle, PRIMARY is read-only.

.

PRIMARY-PASSPHRASE attribute

Sets the secret passphrase (or password) required to authenticate the user identity asserted through attributes of an unsealed client-principal object (in the INITIAL state). This is the same value that you specify for the Password (-P) connection parameter when you connect to an OpenEdge database with the same user identity.

Note: This value has no relation to the domain access code used to seal (or validate the identity of a sealed) client-principal object.

Data type: CHARACTER

Access: Write-only

Applies to: Client-principal object handle

The default value is the Unknown value (?).

The value that you set is the cleartext or encrypted value of the password required to authenticate the user account identity specified using the QUALIFIED-USER-ID attribute. For information on setting an encrypted password value, see ENCRYPT-AUDIT-MAC-KEY( ) method and the topics on encrypted passwords in *OpenEdge Programming Interfaces*.

Attempting to read this attribute returns an error message and sets the ERROR attribute to TRUE on the ERROR-STATUS system handle.

ABL also raises ERROR if you attempt to:

- Assign the Unknown value (?)

- Assign the attribute when the client-principal is sealed and is in the LOGIN, LOGOUT, EXPIRED, or FAILED state (see the LOGIN-STATE attribute entry)

Note:

OpenEdge uses this value to authenticate a user identity only when you call the SECURITY-POLICY:SET-CLIENT( ) method or the SET-DB-CLIENT function on an unsealed client-principal object, which seals the object and sets the identity in a single, OpenEdge-performed user authentication operation.

Use an encrypted value for this attribute especially when you export the unsealed client-principal to a remote authentication service, or to any other ABL session prior to sealing the object. For more information on exporting a client-principal, see EXPORT-PRINCIPAL( ) method.

If your application authenticates the identity, it must also seal the client-principal by calling the SEAL( ) method before using the object to set the identity. However, the SEAL( ) method does not require any setting of this attribute.

OpenEdge does not store the setting of this attribute. Once the client-principal object is sealed, OpenEdge removes all trace of the attribute value from the client-principal.

There is currently no support for specifying a secondary passphrase value in OpenEdge.

See also

ENCRYPT-AUDIT-MAC-KEY( ) method, INITIALIZE( ) method

PRINTER-CONTROL-HANDLE attribute
(Windows only)

The default context for print jobs.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SESSION system handle

The print context is an integer identifier for a set of values that define a printer and setup for that printer in Windows. You can establish a print context using the Print dialog box. The SYSTEM-DIALOG PRINTER-SETUP statement allows you to display the Print dialog box and set a print context. This print context is used by the OUTPUT TO PRINTER statement to direct output to a printer.

If the PRINTER-CONTROL-HANDLE attribute contains zero (0) or the Unknown value (?), the print context is the default print context in Windows as set in the Windows Control Panel. You can assign any integer value to this attribute, but the result of the assignment will be to set the attribute value to 0 and to release any Windows resources related to the previous print context.

PRINTER-HDC attribute

A handle to the current Windows device context for a print job.

Data type: INTEGER

Access: Read-only

Applies to: SESSION system handle

The printer device context handle is the Windows Handle to a Device Context (HDC). The value of the PRINTER-HDC attribute is meaningless when the SESSION:PRINTER-CONTROL-HANDLE has a value of zero (0) or the Unknown value (?). For more information, see the reference entry for the PRINTER-CONTROL-HANDLE attribute.

PRINTER-NAME attribute
(Windows only)

The name of the currently selected printer in Windows platforms, and the Unknown value (?) on other platforms.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SESSION system handle

Use this attribute to set the printer name in the default print context. The value of PRINTER-NAME is the name Windows uses to identify a printer. The specified printer must be defined in the Windows Registry. If the specified printer is not defined in the Windows Registry, the value of PRINTER-NAME is not modified. You must specify network printers in Universal Naming Convention format.

Use the GET-PRINTERS( ) method to get the list of printers currently defined in the Windows Registry.

If you use the SYSTEM-DIALOG PRINTER-SETUP statement to set the printer name, this attribute assumes the modified value.

The printer name can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

PRINTER-PORT attribute
(Windows only)

The currently selected printer port in Windows platforms, and the Unknown value (?) on other platforms.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

PRINTER-PORT assumes the value for printer port that Windows defines. If someone modifies the value using the SYSTEM-DIALOG PRINTER-SETUP command, PRINTER-PORT assumes the modified value.

PRIVATE-DATA attribute

An arbitrary string associated with the handle of an object or widget.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Asynchronous request object handle, BROWSE widget (browse and column), Buffer-field object handle, Buffer object handle, BUTTON widget, Call object handle, COMBO-BOX widget, CONTROL-FRAME widget, Data-relation object handle, Data-source object handle, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, Procedure object handle, ProDataSet object handle, Query object handle, RADIO-SET widget, RECTANGLE widget, SAX-attributes object handle, SAX-reader object handle, SELECTION-LIST widget, Server object handle, Server socket object handle, SLIDER widget, SOAP-header object handle, SOAP-header-entryref object handle, Socket object handle, Stream object handle, SUB-MENU widget, Temp-table object handle, TEXT widget, TOGGLE-BOX widget, WINDOW widget

Use this attribute any way you want. The AVM does not check the value of this attribute.

PROCEDURE-NAME attribute

A string specifying the name of the remote procedure executed to instantiate the specified asynchronous request handle.

Data type: CHARACTER

Access: Read-only

Applies to: Asynchronous request object handle

This name is the same as the *extern-proc-name*, *intern-proc-name*, or VALUE option used to specify the remote procedure executed in the asynchronous RUN statement.

PROCEDURE-TYPE attribute

Specifies whether an external procedure should be invoked as persistent, single-run, or singleton.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Call object handle

This attribute holds a value of "PERSISTENT", "SINGLE-RUN", or "SINGLETON" (not case sensitive), and defaults to the Unknown value (?). Setting this attribute to any other value will result in a run-time error.

When a dynamic invocation returns and PROCEDURE-TYPE is set to "PERSISTENT", the IN-HANDLE attribute will hold a handle to the running persistent procedure. When a dynamic invocation returns and PROCEDURE-TYPE is set to SINGLE-RUN or SINGLETON, the IN-HANDLE attribute will hold a handle to the single-run or singleton procedure.

Setting PROCEDURE-TYPE to "PERSISTENT" will automatically set the PERSISTENT attribute to TRUE. Likewise, setting the PERSISTENT attribute to TRUE will automatically set PROCEDURE-TYPE to "PERSISTENT". Setting the PROCEDURE-TYPE and PERSISTENT attributes to conflicting values, e.g., PERSISTENT as TRUE and PROCEDURE-TYPE as "SINGLE-RUN", will result in a run-time error.

See also

IN-HANDLE attribute, PERSISTENT attribute

PROFILING attribute

Starts measuring timings to profile performance.

Data type: LOGICAL

Access: Readable/Writable

Applies to: PROFILER system handle

The initial value is FALSE.

See also

The following mandatory attributes must be configured prior to profiling, ENABLED attribute, DIRECTORY attribute, DESCRIPTION attribute, and FILE-NAME attribute

PROGRESS-SOURCE attribute
(Character interfaces only)

How an editor widget wraps lines of ABL source code that are longer than the widget's display width.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: EDITOR widget

Set this attribute to TRUE when reading ABL source code into the widget. This preserves the ABL syntax for the ABL compiler.

When set to TRUE, the editor widget splits long lines by putting a tilde (~) and hard return at the end of the display line and continuing the text with column 1 of the next display line. The line wrapping occurs only when a READ-FILE( ) or INSERT-FILE( ) method is used to bring text into the widget. When set to FALSE, the editor widget splits long lines by inserting a HARD-RETURN before the last word and moving the last word onto the next display line. FALSE is the default setting.

PROXY attribute

Indicates whether a procedure handle is a proxy procedure handle.

Data type: LOGICAL

Access: Read-only

Applies to: Procedure object handle

If PROXY is TRUE, the procedure handle is a proxy handle for a remote procedure run with either the PERSISTENT, SINGLE-RUN, or SINGLETON option of the RUN statement in the context of an AppServer.

For more information about the PERSISTENT, SINGLE-RUN, and SINGLETON options, see the RUN statement.

PROXY is always FALSE on the THIS-PROCEDURE handle by definition.

Returns TRUE for a Web service procedure.

PROXY-PASSWORD attribute

Authenticates an application server or Web service client to the HTTP-based proxy server.

Data type: Character

Access: Readable/Writeable

Applies to: SESSION system handle

This attribute corresponds to the `-proxyPassword` startup parameter. If `-proxyPassword` is not specified on the command line, this attribute has the Unknown value (?) until it is set.

This attribute is validated during the CONNECT( ) method (for an application server and a Web service). If PROXY-PASSWORD is invalid, the CONNECT( ) method fails and the AVM issues an error message. If PROXY-USERID **is not** unknown and PROXY-PASSWORD **is** unknown, the application server CONNECT( ) method uses a blank proxy password. If SESSION:PROXY-USERID is unknown, the CONNECT( ) method ignores any value of PROXY-PASSWORD. PROXY-PASSWORD must be a string of up to 512 printable ASCII characters.

See also

PROXY-USERID attribute, *Startup Command and Parameter Reference*, *Develop Web Clients*

PROXY-USERID attribute

Authenticates an application server or Web service client to the HTTP-based proxy server.

Data type: Character

Access: Readable/Writeable

Applies to: SESSION system handle

This attribute corresponds to the `-proxyUserid` startup parameter. If `-proxyUserid` is not specified on the command line, this attribute has the Unknown value (?) until it is set.

This attribute is validated during the CONNECT( ) method (for an application server and a Web service). If PROXY-USERID is invalid, the CONNECT( ) method fails and the AVM issues an error message.

PROXY-USERID must be a string of up to 512 printable ASCII characters, including the space character.

See also

PROXY-USERID attribute, *Startup Command and Parameter Reference*, *Develop Web Clients*

PUBLIC-ID attribute

This attribute returns the public ID of the external DTD from which an XML document was generated.

Data type: CHARACTER

Access: Read-only

Applies to: X-document object handle

PUBLISHED-EVENTS attribute

A comma-separated list of ABL named events published by a particular procedure. Returns the empty string for a Web service procedure.

Note: ABL named events are completely different from the key function, mouse, widget, and direct manipulation events described in the Handle-based Object Events Reference. They are also different from the class events described in the Class Events Reference.

Data type: CHARACTER

Access: Read-only

Applies to: Procedure object handle, SOURCE-PROCEDURE system handle, TARGET-PROCEDURE system handle, THIS-PROCEDURE system handle

ABL builds PUBLISHED-EVENT lists as the compiler encounters PUBLISH statements-specifically, PUBLISH statements that specify the event name as a quoted string and not as a CHARACTER variable expression, and that do not use the FROM option.

Note: PUBLISHED-EVENTS lists do not contain signatures of their named events. ABL assumes that the subscriber to a named event knows its signature.

QUALIFIED-USER-ID attribute

Sets or reads the values of the USER-ID attribute (user name) and DOMAIN-NAME attribute separated by the `'@'` domain delimiter. This value represents a fully qualified user identity that you can use to establish identity in ABL sessions or database connections using the SECURITY-POLICY:SET-CLIENT( ) method or the SET-DB-CLIENT function. This user ID is then used to authorize such actions as accessing resources (run-time permissions checking) and providing an auditing identity, among other things. For a multi-tenant database connection, the user's identity also establishes the user's database tenant organization.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

You can write a character expression to this attribute using one of the formats in the following table, where *`user-name`* is a specified value for the USER-ID attribute and *`domain-name`* is a specified value for the DOMAIN-NAME attribute.

Table 94: Formats for assigning the QUALIFIED-USER-ID attribute

This format...	Writes...
`""`	The blank user in the blank domain
`"@"`	The blank user in the blank domain
`"@`*`domain-name`*`"`	The blank user in the specified domain
`"`*`user-name`*`"`	The specified user in the blank domain

This format...	Writes...
"*user-name*@"	The specified user in the blank domain
"*user-name*@*domain-name*"	The specified user in the specified domain

When you read this attribute, it can return the values shown in the following table.

Table 95: Values returned when reading the QUALIFIED-USER-ID attribute

This value is returned...	For...
""	The blank user in the blank domain
"@*domain-name*"	The blank user in the specified domain
"*user-name*"	The specified user in the blank domain
"*user-name*@*domain-name*"	The specified user in the specified domain
Unknown value (?)	USER-ID or DOMAIN-NAME attributes that have not yet been set

Once the client-principal object is sealed, this attribute is read-only, and attempting to write to it raises a run-time error.

ABL also raises ERROR if you attempt to assign:

- The Unknown value (?)
- Any character expression that does not conform to the allowable qualified user ID forms shown in Table 94: Formats for assigning the QUALIFIED-USER-ID attribute
- A character expression when the client-principal object is sealed and in the LOGIN, LOGOUT, EXPIRED, or FAILED state (see the LOGIN-STATE attribute entry)

See also

INITIALIZE( ) method, DOMAIN-NAME attribute, USER-ID attribute

QUERY attribute

The handle of the query connected to a browse widget, a buffer object, a data-relation object, or a data-source object.

Data type: HANDLE

Access: Readable/Writable

Applies to: BROWSE widget, Buffer object handle, Data-relation object handle, Data-source object handle

If you change the value of a browse's QUERY attribute, you connect the browse to a different query, which contains a different set of records.

For a browse query in Windows platforms:

- The original query and the new query do not need to have the same underlying database fields.

- If the query is changed for a dynamic browse, the browse columns are removed. You should add new columns with the ADD-CALC-COLUMN, ADD-COLUMNS-FROM, and ADD-LIKE-COLUMN methods.
- If the query is changed for a static browse and the underlying fields are the same, the columns are not removed. However, if the underlying fields are not the same, the columns are removed. The columns are also removed if the QUERY attribute is set to the Unknown value (?). You should add new columns with the ADD-CALC-COLUMN, ADD-COLUMNS-FROM, and ADD-LIKE-COLUMN methods.
- Also, a query can now be attached to a static browse that was defined without the optional DISPLAY phrase.

For a browse query on Character Mode platforms:

- If the original query has database tables, the new query must have database tables. The new query can have different buffers as long as they correspond to the same database tables.
- If the original query has temp-tables, the new query must have temp-tables, not work tables.
- The original query and the new query must have the same number of tables.

For a buffer object, this attribute returns the handle to the query currently associated with the buffer (if any). If the buffer does not have an associated query, this attribute returns the Unknown value (?). This attribute is also read-only for a buffer object.

For a data-relation object, this attribute returns the handle to the default dynamic query for a child buffer in the relation. This automatically generated query expresses the relation between parent and child temp-tables, and lets you navigate the child records. This handle cannot be set, and the query cannot be modified except by using the WHERE-STRING attribute.

For a data-source object, this attribute associates a query with a dynamic data-source object. To disassociate the query and data-source object, set this attribute to the Unknown value (?). You can also use the FILL-WHERE-STRING attribute to override the WHERE clause in the query.

See also

FILL-WHERE-STRING attribute, WHERE-STRING attribute

QUERY-CLOSE( ) method

Closes a query object.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
QUERY-CLOSE ( )
```

Note: A QUERY-CLOSE does not invalidate a previous QUERY-PREPARE.

QUERY-OFF-END attribute

Indicates whether a query is positioned off either end of its result list (that is, either before the first record or after the last record).

Data type: LOGICAL

Access: Read-only

Applies to: Query object handle

The QUERY-OFF-END attribute corresponds to the QUERY-OFF-END function.

Note: ABL also provides an OFF-END event for when a query on a ProDataSet temp-table buffer is positioned past the last row. You can use this event to retrieve additional data source rows to add at the bottom of a ProDataSet temp-table (for example, in batches when there are too many data source rows to retrieve at one time). The OFF-END event is similar to the QUERY-OFF-END attribute, which is set to TRUE whenever the associated query object is positioned past the last row. The difference is that you must test the QUERY-OFF-END attribute for this condition at a specific place in your application code, whereas the OFF-END event procedure executes like a trigger whenever the event occurs. For more information about the OFF-END event on a ProDataSet temp-table query, see the ProDataSet events.

See also

QUERY-OFF-END function

QUERY-OPEN( ) method

Opens a query object.

Note: You must perform QUERY-PREPARE on a query object before you perform QUERY-OPEN on it.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
QUERY-OPEN ( )
```

Once you perform QUERY-PREPARE on a query object, you can perform QUERY-OPEN on it multiple times as long as you do not reperform QUERY-PREPARE. Once you reperform QUERY-PREPARE, you must reperform QUERY-OPEN.

Note: The QUERY-PREPARE and QUERY OPEN methods correspond to the OPEN QUERY statement.

QUERY-PREPARE( ) method

Compiles a predicate (query condition).

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
QUERY-PREPARE ( predicate-expression )
```

predicate-expression

A CHARACTER expression that evaluates to an OPEN QUERY ... FOR EACH statement without the OPEN QUERY .... You can also use a field phrase.

The QUERY-PREPARE method corresponds to the OPEN QUERY statement's compilation phase. To open the query object, use the QUERY-OPEN method.

If the QUERY-PREPARE method encounters an error, it returns FALSE and generates an error message, but does not raise ERROR. If you use the QUERY-PREPARE method in a statement that uses the NO-ERROR option and an error occurs, the QUERY-PREPARE method returns FALSE and diverts the error to the ERROR-STATUS system handle. In either instance, ERROR-STATUS:ERROR returns FALSE. You can get information on the error through the GET-MESSAGE method of the ERROR-STATUS system handle, as usual.

Note: The QUERY-PREPARE method is compatible with indexed reposition of queries with joins. In *predicate-expression*, just include the INDEXED-REPOSITION option. For more information on the INDEXED-REPOSITION option, see the reference entry for the OPEN QUERY statement.

The following are examples:

```
hQuery:QUERY-PREPARE("FOR EACH Customer WHERE Customer.CustNum < 9").
```

```
hQuery:QUERY-PREPARE(my-predicate).
```

```
hQuery:QUERY-PREPARE("FOR EACH Customer WHERE Customer.CustNum > " +
  STRING(my-integer)).
```

Unless explicitly specified otherwise in the *predicate-expression*, the default record lock type for the QUERY-PREPARE method is NO-LOCK.

Like the FOR statement, the QUERY-PREPARE method *predicate-expression* supports the BREAK and BY options for sorting and accumulating data by break groups.

```
hQuery:QUERY-PREPARE("FOR EACH Customer WHERE Customer.CustNum > 50
  BREAK BY Customer.SalesRep BY Customer.Country").
```

To test whether a break group has changed, you can use the FIRST-OF( ) method and LAST-OF( ) method of the query object handle.

The QUOTER function can be used to wrap character values with internal quotes, as in this example:

```
hQuery:QUERY-PREPARE("FOR EACH Customer WHERE Customer.Name = " +
  QUOTER(my-name)).
```

Note: The NAME attribute of the Temp-table object handle is writeable for dynamic and AVM-generated temp-tables. You might need to update a *predicate-expression* that references a renamed temp-table with new strings using the new table name.

See also

OPEN QUERY statement

QUIT attribute

Indicates that a QUIT condition was returned from the application server as a result of processing the specified asynchronous request. Returns FALSE for an asynchronous request made on a Web service.

Data type: LOGICAL

Access: Read-only

Applies to: Asynchronous request object handle

If the COMPLETE attribute is FALSE, the value of this attribute is the Unknown value (?). When the PROCEDURE-COMPLETE event is processed, this attribute is set to TRUE before the event procedure or method is executed, if the remote request returned with an unhandled QUIT condition; otherwise, it is set to FALSE.

RADIO-BUTTONS attribute

The label and value associated with each radio button in a radio set.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: RADIO-SET widget

You can set this attribute to a comma-separated list containing the label/value pairs associated with each button. Each label and each value should be followed by a comma, as in "label1,value1,label2,value2,...label*n*,value*n*".

RAW-TRANSFER( ) method

Copies data to or from a buffer object with no interpretation. This method works like the RAW-TRANSFER statement.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
RAW-TRANSFER ( to-mode , handle-expression )
```

to-mode

A logical specifying the direction of the data transfer. When *to-mode* is TRUE, data is transferred from the buffer object handle to *handle-expression*. When *to-mode* is FALSE, data is transferred from *handle-expression* to the buffer object handle.

handle-expression

An expression that evaluates to the handle of either a buffer or a buffer field.

When using the RAW-TRANSFER statement to copy from a buffer object that contains a BLOB or CLOB field, the AVM skips the BLOB or CLOB field and stores the Unknown value (?) in the BLOB or CLOB field of the target buffer object.

See also

RAW-TRANSFER statement

READ( ) method (Socket)

Reads data from the socket.

Return type: LOGICAL

Applies to: Socket object handle

Syntax

```
READ ( buffer , position , bytes-to-read , [ mode ] )
```

buffer

A MEMPTR expression that identifies where the data which is read from the socket should be stored.

position

An integer expression greater than 0 that indicates the starting byte position within *buffer* into which information should be written.

bytes-to-read

An integer expression that specifies the number of bytes to be read from the socket.

mode

An optional integer expression that specifies how *bytes-to-read* should be interpreted. The following table shows the valid values for this parameter. The default value is READ-EXACT-NUM (2).

Table 96: Valid read modes for the READ( ) method

Compiler constant	Value	Description
READ-AVAILABLE	1	The READ( ) method will block until at least one byte has been read on the socket. It will read up to *bytes-to-read* bytes.
READ-EXACT-NUM	2	The READ( ) method will block until *bytes-to-read* bytes have been read from the socket.

READ( ) returns TRUE if the read operation succeeded normally and returns FALSE otherwise. An error can occur if:

- The position parameter is not greater than 0.
- Amount of information requested to read exceeds the size of buffer.
- Reading from the socket fails.

This read statement is a blocking read. If *mode* is READ-EXACT-NUM, this method returns when it has either read the requested number of bytes from the socket or an error occurs. If *mode* is READ-AVAILABLE, this method returns when it has read as many bytes as are currently available on the socket, up to the requested number of bytes, or an error occurs.

If the READ( ) method succeeds, the variable buffer contains the data which is read from the socket. It is possible that the socket will not contain the specified number of bytes of data which were requested. The BYTES-READ attribute can be used to determine the number of bytes read from the socket.

This method expects *buffer* to identify a MEMPTR variable which already has a region of memory associated with it. The developer must call the SET-SIZE statement to allocate memory and associate it with a MEMPTR variable. It is the responsibility of the developer to free this memory, also via the SET-SIZE statement. The READ method will fail if the size of *buffer* is less than *bytes-to-read*.

READ-FILE( ) method

Clears an editor widget, reads the contents of a specified text file into the widget, and sets the widget's MODIFIED attribute to FALSE.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
READ-FILE ( filename )
```

filename

A character-string expression equal to the full or relative pathname of a file. If you specify a relative pathname, the READ-FILE( ) method searches PROPATH to find the file. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

If the operation is successful, the method returns TRUE.

In Windows, this method interprets a carriage return character followed by a line feed character as a text line terminator. In all other interfaces, this method interprets a carriage return character as a text line terminator.

READ-JSON( ) method

Reads a specified JSON string, `Progress.Json.ObjectModel.JsonArray` object, or `Progress.Json.ObjectModel.JsonObject` object into a corresponding ProDataSet, a temp-table, or a temp-table buffer object.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

Syntax

```
READ-JSON ( source-type
  , { file | memptr | handle | longchar | JsonArray | JsonObject }
  [ , read-mode ] )
```

source-type

A CHARACTER expression that specifies the source JSON string type. Valid values are `"FILE"`, `"MEMPTR"`, `"HANDLE"`, `"LONGCHAR"`, `"JsonArray"`, and `"JsonObject"`.

All values except `"JsonArray"` and `"JsonObject"` specify a source for a JSON string.

`"JsonArray"` is only a valid *source-type* for invoking READ-JSON( ) on a temp-table or buffer object handle. If set to `"JsonArray"` for a ProDataSet object handle, the READ-JSON( ) method generates an error message and returns a value of FALSE.

file

A CHARACTER expression that specifies the name of a file. You can specify an absolute pathname or one relative to the current working directory. The AVM verifies that the file exists and is accessible.

memptr

A MEMPTR variable that contains the JSON string in memory. The size of the MEMPTR variable must match the size of the JSON string.

handle

A HANDLE variable that specifies the WEB-CONTEXT system handle.

This method reads a JSON string from the WebSpeed Transaction Server. The method verifies that the JSON string was posted to the WebSpeed Transaction Server by checking that the handle's IS-JSON attribute is YES. The method also verifies that ABL is running in a WebSpeed environment.

longchar

A LONGCHAR variable that contains the JSON string in memory.

JsonArray

A reference to a Progress.Json.ObjectModel.JsonArray object.

This *JsonArray* object must fit one of the valid patterns for a temp-table object. If any part of the *JsonArray* object does not fit one of the accepted patterns, the READ-JSON( ) method generates an error message and returns FALSE. If the *JsonArray* object fits an accepted pattern, but contains fields whose names do not match the existing temp-table schema, the mismatched fields are ignored.

JsonObject

A reference to a Progress.Json.ObjectModel.JsonObject object.

This *JsonObject* object must fit one of the valid patterns for a ProDataSet or temp-table object. If any part of the *JsonObject* object does not fit one of the accepted patterns, the READ-JSON( ) method generates an error message and returns FALSE. If the *JsonObject* object fits an accepted pattern, but contains tables whose names do not match the existing ProDataSet schema, or contains fields whose names do not match the existing temp-table schema, the mismatched tables or fields are ignored.

read-mode

An optional CHARACTER expression that specifies the mode in which this method reads data from the JSON source into a temp-table or a ProDataSet member buffer. The expression must evaluate to "APPEND", "EMPTY", "MERGE", or "REPLACE". The default value is "MERGE".

The following table lists the READ-JSON( ) method modes for reading data.

Table 97: READ-JSON( ) method read modes

When the mode is . . .	The READ-JSON( ) method . . .
"APPEND"	Reads data from the JSON source into the ProDataSet or temp-table object by adding new records to the existing records, without performing any record comparisons. If a record from the JSON source exists in the object (that is, it results in a duplicate unique key conflict), the method generates an error message and returns FALSE.
"EMPTY"	Empties the contents of the ProDataSet or temp-table object before reading in data from the JSON source.

When the mode is . . .	The READ-JSON( ) method . . .
"MERGE"	Reads data from the JSON source into the ProDataSet or temp-table object by merging new records with existing records in the table. If a record from the JSON source exists in the object (that is, it results in a duplicate unique key conflict), the method does not replace the existing record. If the record from the JSON source does not exist in the object, the method creates a new record.
"REPLACE"	Reads data from the JSON source into the ProDataSet or temp-table object by merging new records with existing records in the table. If the record from the JSON source exists in the object (that is, it results in a duplicate unique key conflict), the method replaces the existing record with the new record. If the record from the JSON source does not exist in the object, the method creates a new record.

Notes

- For a dynamic ProDataSet or temp-table that is in the CLEAR state, the AVM infers the object's schema from the data in the JSON value. If a dynamic temp-table is not in the PREPARED or CLEAR state, the method generates an error message and returns FALSE. For more information about inferring schema from a JSON source, see *Use JSON with ABL Applications*.
- For a static ProDataSet or temp-table, the serialize name or object name must match the name found in the JSON source. If the names do not match, the method generates an error message and returns FALSE. The AVM ignores any columns in the JSON source that do not map to temp-table columns. If you use the `SERIALIZE-NAME` option in the `DEFINE DATASET` or `DEFINE TEMP-TABLE` statement, the AVM uses that name for matching, rather than the ABL object name.
- If a JSON source contains ProDataSet before-image data, the `READ-JSON( )` method populates the after-table and before-table data for the ProDataSet.
- You cannot call `READ-JSON( )` on a database buffer handle.
- `READ-JSON( )` can deserialize class-based objects from a JSON source. See WRITE-JSON( ) method for a detailed list of restrictions that apply to any class-based objects that are to be written to and from JSON.
- When executing the method, there is no guarantee as to the contents of the buffers involved after the method finishes executing.
- Binary types, including BLOB, RAW, and ROWID are base64-encoded/decoded. See Data type mapping in *Use JSON with ABL Applications* for more information.

Examples

The following code example creates a dynamic ProDataSet object from an empty ProDataSet handle, creates the objects schema by inference from the specified JSON string, and populates the temp-tables with records from the specified JSON document:

```
DEFINE VARIABLE cSourceType AS CHARACTER NO-UNDO.
DEFINE VARIABLE cReadMode   AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFile       AS CHARACTER NO-UNDO.
DEFINE VARIABLE lRetOK      AS LOGICAL   NO-UNDO.
DEFINE VARIABLE hDSet       AS HANDLE    NO-UNDO.

CREATE DATASET hDSet.
ASSIGN
  cSourceType = "file"
  cFile       = "dset.json"
  cReadMode   = "empty".

lRetOK = hDSet:READ-JSON(cSourceType, cFile, cReadMode).
```

The following code example creates a dynamic temp-table object from an empty temp-table handle, creates the object's schema by inference from the specified JSON string, and populates the temp-table with records from the same string:

```
DEFINE VARIABLE cSourceType AS CHARACTER NO-UNDO.
DEFINE VARIABLE cReadMode   AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFile       AS CHARACTER NO-UNDO.
DEFINE VARIABLE lRetOK      AS LOGICAL   NO-UNDO.
DEFINE VARIABLE httCust     AS HANDLE    NO-UNDO.
CREATE TEMP-TABLE httCust.

ASSIGN
  cSourceType = "file"
  cFile       = "ttcust.json"
  cReadMode   = "empty".

lRetOK = httCust:READ-JSON(cSourceType, cFile, cReadMode).
```

See also

IS-JSON attribute, Progress.Json.ObjectModel.JsonArray class, Progress.Json.ObjectModel.JsonObject class, WEB-CONTEXT system handle, WRITE-JSON( ) method , SERIALIZE-ROW( ) method

READ-ONLY attribute

Indicates whether an object is write-protected.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), Buffer-field object handle, EDITOR widget, FILL-IN widget, MENU-ITEM widget

If the READ-ONLY attribute of an editor or fill-in widget is TRUE, the widget cannot be enabled for input and its screen value cannot be changed from the user interface.

If the READ-ONLY attribute for a browse widget is TRUE, you cannot update editable cells. If it is set to false, then the ability to edit cells is restored. This functionality only applies to columns that have been enabled in the DEFINE BROWSE statement.

For combo-box browse columns, this attribute is read-only and always returns FALSE.

If the READ-ONLY attribute of a menu item is TRUE, the menu item cannot be chosen. You can set this attribute for a menu item only before the widget is realized.

The READ-ONLY attribute has no effect on the appearance of a widget. If an editor or menu item is insensitive, it is grayed out in some environments. Use the READ-ONLY attribute instead to make the widget insensitive without being grayed out.

READ-XML( ) method

Reads an XML document into a ProDataSet, temp-table, or temp-table buffer object. You can read data, schema, or both.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

Syntax

```
READ-XML ( source-type
  , { file | memptr | handle | longchar }
  , read-mode , schema-location , override-default-mapping
  [ , field-type-mapping [ , verify-schema-mode ] ] )
```

source-type

A CHARACTER expression that specifies the source XML document type. Valid values are `"FILE"`, `"MEMPTR"`, `"HANDLE"`, and `"LONGCHAR"`.

file

A CHARACTER expression that specifies the name of a file. You can specify an absolute pathname, a relative pathname (based on the current working directory), or a URL pathname. Valid URL protocols include FILE and HTTP (the HTTPS protocol is not supported). The AVM verifies that the file exists and is accessible The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

memptr

A MEMPTR variable that contains the XML document text in memory. The size of the MEMPTR variable must match the size of the XML document text.

handle

A `WEB-CONTEXT` system handle, X-document object handle, or X-noderef object handle.

For a WEB-CONTEXT system handle, the READ-XML( ) method reads an XML document from the WebSpeed transaction server. The method verifies that the XML document was posted to the WebSpeed transaction server (that is, the value of the IS-XML attribute for the handle is YES), and that ABL is running in a WebSpeed environment.

longchar

A LONGCHAR variable that contains the XML document text in memory.

read-mode

A CHARACTER expression that specifies the mode in which the READ-XML( ) method reads data from the XML document into a temp-table or ProDataSet member buffer. The expression must evaluate to "APPEND", "EMPTY", "MERGE", or "REPLACE". The default value is "MERGE".

The following table lists the READ-XML( ) method modes for reading data.

Table 98: READ-XML( ) method read modes

When the mode is . . .	The READ-XML( ) method . . .
APPEND	Reads data from the XML document into the ProDataSet or temp-table object by adding new records to the existing records, without performing any record comparisons. If a record from the XML document exists in the object (that is, it results in a duplicate unique key conflict), the method generates an error message and returns FALSE.
EMPTY	Empties the contents of the ProDataSet or temp-table object before reading in data from the XML document.
MERGE	Reads data from the XML document into the ProDataSet or temp-table object by merging new records with existing records in the table. If a record from the XML document exists in the object (that is, it results in a duplicate unique key conflict), the method does not replace the existing record. If the record from the XML document does not exist in the object, the method creates a new record.
REPLACE	Reads data from the XML document into the ProDataSet or temp-table object by merging new records with existing records in the table. If the record from the XML document exists in the object (that is, it results in a duplicate unique key conflict), the method replaces the existing record with the new record. If the record from the XML document does not exist in the object, the method creates a new record.

schema-location

A CHARACTER expression that specifies the name of an external XML Schema file to use in creating or verifying the object's schema when reading in the XML document. You can specify an absolute pathname, a relative pathname (based on the current working directory), or a URL pathname. Valid URL protocols include FILE and HTTP (the HTTPS protocol is not supported). The AVM verifies that the file exists and is accessible. When specified, the AVM ignores any schema defined or referenced in the source XML Document. The pathname can contain Unicode characters.

If you specify the empty string ("") or the Unknown value (?), the AVM creates or verifies the object's schema using any XML Schema defined or referenced in the XML document.

override-default-mapping

A LOGICAL expression where TRUE directs the AVM to override the default mapping between XML Schema string and binary data types and ABL data types when creating an ABL temp-table schema from an XML Schema. The default value is FALSE.

The XML Schema string data type maps to the ABL CHARACTER data type by default, and the XML Schema base64Binary and hexBinary data types map to the ABL RAW data type by default. If you specify TRUE, the READ-XML( ) method creates a temp-table schema with CLOB and BLOB fields instead of CHARACTER and RAW fields.

If you specify the Unknown value (?), the method uses the default value of FALSE.

field-type-mapping

An optional CHARACTER expression that evaluates to a comma-delimited list of field name, data type pairs using the following syntax:

```
"field-name-1,data-type-1[,field-name-n,data-type-n]..."
```

This option allows you to specify the ABL data type for a specific field from the XML Schema.

field-name

A CHARACTER expression that evaluates to the name of the specified field. For a ProDataSet object, you must qualify the field name with the buffer name from the XML Schema. That is, *buffer-name.field-name*.

data-type

A CHARACTER expression that evaluates to the data type of the specified field. The data type must be a valid ABL data type, and it must be compatible with the XML Schema type based on the ABL XML data type mapping rules. For example, any XML Schema type can be mapped to an ABL CHARACTER or CLOB, but an XML Schema dateTime can be mapped only to an ABL DATE, DATETIME or DATETIME-TZ.

If you specify the Unknown value (?), the method uses the default data type mapping. For more information about the ABL XML data type mapping rules, see *Use XML with ABL Applications*.

verify-schema-mode

An optional CHARACTER expression that specifies the mode in which the READ-XML( ) method verifies any XML Schema against existing ABL schema. The expression must evaluate to "IGNORE", "LOOSE", or "STRICT". The default value is "LOOSE".

Note: For a dynamic temp-table or ProDataSet member buffer that does not have ABL schema (that is, the object is in the CLEAR state), this option is ignored.

The following table lists the READ-XML( ) method schema verification modes.

Table 99: READ-XML( ) method schema verification modes

When the mode is . . .	The READ-XML( ) method . . .
IGNORE	Ignores any XML Schema specified in *schema-location*, or defined in the XML Document.
LOOSE for temp-table objects	• Matches temp-table columns by name. The data type and extent of the column in the XML Schema must match those for the matching column in the temp-table. Other field attributes in the XML Schema are ignored. • The XML Schema may be a subset or superset of the temp-table schema. Any columns that are in the XML Schema but not in the temp-table are ignored. Any columns that are in the temp-table, but not in the XML Schema, are ignored.
LOOSE for ProDataSet objects	• Matches temp-tables and columns by name. The data type and extent of the column in the XML Schema must match those for the matching column in the temp-table. Other field attributes in the XML Schema are ignored. • Data relationships are matched by parent buffer and child buffer names. For every data relationship in the XML Schema that matches a data-relation in the ProDataSet, the field mapping between the parent and child buffers must match. • The XML Schema may be a subset or superset of the ProDataSet schema. Any temp-tables, columns, or data-relations that are in the ProDataSet, but not in the XML Schema, are ignored. For a dynamic ProDataSet object, the method adds temp-tables and data-relations to the object when the temp-tables and data-relations are defined in the XML Schema, but are not members of the ProDataSet. Fields are not added to existing temp-tables. For a static ProDataSet object, any temp-tables or data-relations that are in the XML Schema, but not in the ProDataSet, are ignored.

When the mode is . . .	The READ-XML( ) method . . .
STRICT for temp-table objects	• Matches temp-table columns by name. The data type and extent of the column in the XML Schema must match those for the matching column in the temp-table. Other field attributes in the XML Schema are ignored. • The XML Schema must define a field for every temp-table field, and there can be no extra fields in the XML Schema that are not in the temp-table schema.
STRICT for ProDataSet objects	• Matches temp-tables and columns by name. There must be a temp-table defined in the XML Schema for each table of the ProDataSet. There can be no tables defined in the XML Schema that are not in the ProDataSet schema. • There must be a field defined in the XML Schema for each field in the temp-table, with the same data type and extent, and there can be no fields defined in the XML Schema that are not in the temp-table schema. • There must also be a data relationship defined in the XML Schema for every data-relation in the ProDataSet, and there can be no data relationships defined in the XML Schema that are not defined in the ProDataSet. The field mapping between the parent and child buffers must match.

If you specify the Unknown value (?), the method uses the default value of LOOSE.

If the XML Schema verification fails, the method generates an error message indicating the XML Schema element that caused the failure and returns FALSE.

Notes

- If the ProDataSet or temp-table object does not have a schema (that is, the object is dynamic and in the CLEAR state), the AVM creates the schema from either the XML Schema file specified in *schema-location*, or the XML Schema defined or referenced in the XML document. If a dynamic temp-table is not in the PREPARED or CLEAR state, the method generates an error and returns FALSE.
- ABL has a pair of attributes with overlapping purposes in how the AVM reads XML data, SERIALIZE-NAME and XML-NODE-NAME. Refer to the attribute entries for a full description of their interaction.
- If the ProDataSet or temp-table object already has a schema (that is, the object is static, or the temp-tables are in the PREPARED state), the AVM verifies any XML Schema specified by *schema-location*, or defined or referenced in the XML document, against the object's schema, unless the *verify-schema-mode* is "IGNORE".
- If the AVM cannot identify any XML Schema for the ProDataSet or temp-table object, (that is, *schema-location* is the empty string ("") or the Unknown value (?) and the XML document does not define or reference a schema, the AVM infers the schema from the data in the XML document.
- For more information about creating schema from XML Schema, verifying XML Schema, or inferring schema from XML document text, see *Use XML with ABL Applications*.
- The XML document can also contain before-image table data associated with a ProDataSet object. If the XML document data is in the Microsoft DiffGram format, the method reads the before-image data as well. In this case, if the ProDataSet or temp-table object is static and it does not have a before-image table defined, the method generates an error and returns FALSE. If the ProDataSet or temp-table object is dynamic, the method creates the before-image table automatically.

- During the read operation, the AVM does not respond to ProDataSet events, and it does not track changes to the data in the ProDataSet or temp-table object (that is, it does not update the before-image tables) unless the XML document data is in the Microsoft DiffGram format.
- You cannot read an XML document into a database buffer.
- When executing the method, there is no guarantee as to the contents of the buffers involved after the method finishes executing.
- Binary types, including BLOB, RAW, and ROWID are base64-encoded/decoded, unless the XML-DATA-TYPE attribute is hexBinary. If `XML-DATA-TYPE` is hexBinary, then BLOB, RAW, and ROWID are hex-encoded/decoded. See Default mappings from other XML Schema data types to ABL data types in *Use XML with ABL Applications* for more information.

Examples

The following code example creates a dynamic ProDataSet object from an empty ProDataSet handle, creates the object's schema from the specified XML Schema file, and populates the temp-tables with records from the specified XML document:

```
DEFINE VARIABLE cSourceType             AS CHARACTER NO-UNDO.
DEFINE VARIABLE cReadMode               AS CHARACTER NO-UNDO.
DEFINE VARIABLE lOverrideDefaultMapping AS LOGICAL   NO-UNDO.
DEFINE VARIABLE cFile                   AS CHARACTER NO-UNDO.
DEFINE VARIABLE cSchemaLocation         AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFieldTypeMapping       AS CHARACTER NO-UNDO.
DEFINE VARIABLE cVerifySchemaMode       AS CHARACTER NO-UNDO.
DEFINE VARIABLE lRetOK                  AS LOGICAL   NO-UNDO.
DEFINE VARIABLE hDSet                   AS HANDLE    NO-UNDO.

CREATE DATASET hDSet.
ASSIGN
  cSourceType             = "file"
  cFile                   = "dset.xml"
  cReadMode               = "empty"
  cSchemaLocation         = "cust-ord-inv.xsd"
  lOverrideDefaultMapping = ?
  cFieldTypeMapping       = ?
  cVerifySchemaMode       = ?.

lRetOK = hDSet:READ-XML(cSourceType, cFile, cReadMode, cSchemaLocation,
  lOverrideDefaultMapping, cFieldTypeMapping, cVerifySchemaMode).
```

The following code example creates a dynamic temp-table object from an empty temp-table handle, creates the object's schema from the specified XML Schema file, and populates the temp-table with records from the specified XML document:

```
DEFINE VARIABLE cSourceType             AS CHARACTER NO-UNDO.
DEFINE VARIABLE cReadMode               AS CHARACTER NO-UNDO.
DEFINE VARIABLE lOverrideDefaultMapping AS LOGICAL   NO-UNDO.
DEFINE VARIABLE cFile                   AS CHARACTER NO-UNDO.
DEFINE VARIABLE cSchemaLocation         AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFieldTypeMapping       AS CHARACTER NO-UNDO.
DEFINE VARIABLE cVerifySchemaMode       AS CHARACTER NO-UNDO.
DEFINE VARIABLE lRetOK                  AS LOGICAL   NO-UNDO.
DEFINE VARIABLE httCust                 AS HANDLE    NO-UNDO.

CREATE TEMP-TABLE httCust.
ASSIGN
  cSourceType             = "file"
  cFile                   = "ttcust.xml"
  cReadMode               = "empty"
  cSchemaLocation         = "ttcust.xsd"
  lOverrideDefaultMapping = ?
  cFieldTypeMapping       = ?
  cVerifySchemaMode       = ?.

lRetOK = httCust:READ-XML(cSourceType, cFile, cReadMode, cSchemaLocation,
  lOverrideDefaultMapping, cFieldTypeMapping, cVerifySchemaMode).
```

See also

IS-XML attribute, READ-XMLSCHEMA( ) method, SERIALIZE-NAME attribute, WEB-CONTEXT system handle, WRITE-XML( ) method, WRITE-XMLSCHEMA( ) method, X-document object handle, X-noderef object handle, XML-NODE-NAME attribute

READ-XMLSCHEMA( ) method

Reads XML Schema from an XML document and uses that schema to either create a schema for a ProDataSet or temp-table object, or verify existing schema in a ProDataSet, temp-table, or temp-table buffer object.

Note: The XML document must be an XML Schema written in the XML Schema Definition (XSD) language in the 2001 XML Schema namespace (http://www.w3.org/2001/XMLSchema). Non-OpenEdge applications might use XSD in many ways. The `READ-XMLSCHEMA( )` method attempts to parse any XSD elements that form an obvious relational structure into ABL temp-tables and ProDataSets. However, the method might not be able to handle every XSD element generated by a non-OpenEdge source. The closer a particular XSD element conforms to what the `WRITE-XMLSCHEMA( )` method creates, the more likely that the `READ-XMLSCHEMA( )` method will succeed.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

Syntax

```
READ-XMLSCHEMA ( source-type
  , { file | memptr | handle | longchar }
  , override-default-mapping
  [ , field-type-mapping [ , verify-schema-mode ] ] )
```

source-type

A CHARACTER expression that specifies the source XML document type. Valid values are: "FILE", "MEMPTR", "HANDLE", and "LONGCHAR".

file

A CHARACTER expression that specifies the name of an XML Schema file. You can specify an absolute pathname, a relative pathname (based on the current working directory), or a URL pathname. Valid URL protocols include FILE and HTTP (the HTTPS protocol is not supported). The AVM verifies that the file exists and is accessible. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

memptr

A MEMPTR variable that contains the XML Schema document text. The size of the MEMPTR variable must match the size of the XML document text.

handle

A WEB-CONTEXT system handle, X-document object handle, or X-noderef object handle.

For a WEB-CONTEXT system handle, the READ-XMLSCHEMA( ) method reads an XML Schema document from the WebSpeed transaction server. The method verifies that the XML document was posted to the WebSpeed transaction server (that is, the value of the IS-XML attribute for the handle is YES), and that ABL is running in a WebSpeed environment.

longchar

A LONGCHAR variable that contains the XML Schema document text in memory.

override-default-mapping

A LOGICAL expression where TRUE directs the AVM to override the default mapping between XML Schema string and binary data types and ABL data types when creating an ABL temp-table schema from an XML Schema. The default value is FALSE.

The XML Schema string data type maps to the ABL CHARACTER data type by default, and the XML Schema base64Binary and hexBinary data types map to the ABL RAW data type by default. If you specify TRUE, the READ-XMLSCHEMA( ) method creates a temp-table schema with CLOB and BLOB fields instead of CHARACTER and RAW fields.

If you specify the Unknown value (?), the method uses the default value of FALSE.

field-type-mapping

An optional CHARACTER expression that evaluates to a comma-delimited list of field name, data type pairs using the following syntax:

```
"field-name-1,data-type-1[,field-name-n,data-type-n]..."
```

This option allows you to specify the ABL data type for a specific field from the XML Schema.

field-name

A CHARACTER expression that evaluates to the name of the specified field. For a ProDataSet object, you must qualify the field name with the buffer name from the XML Schema. That is, *buffer-name.field-name*.

data-type

A CHARACTER expression that evaluates to the data type of the specified field. The data type must be a valid ABL data type, and it must be compatible with the XML Schema type based on the ABL XML data type mapping rules. For example, any XML Schema type can be mapped to an ABL CHARACTER or CLOB, but an XML Schema dateTime can be mapped only to an ABL DATE, DATETIME or DATETIME-TZ.

If you specify the Unknown value (?), the method uses the default data type. For more information about the ABL XML data type mapping rules, see *Use XML with ABL Applications*.

verify-schema-mode

An optional CHARACTER expression that specifies the mode in which the READ-XMLSCHEMA( ) method verifies any XML Schema against existing ABL schema. The expression must evaluate to `"LOOSE"` or `"STRICT"`. The default value is `"LOOSE"`.

Note: For a dynamic temp-table or ProDataSet member buffer that does not have ABL schema (that is, the object is in the CLEAR state), this option is ignored.

The following table lists the READ-XMLSCHEMA( ) method schema verification modes.

Table 100: READ-XMLSCHEMA( ) method verification modes

When the mode is . . .	The READ-XMLSCHEMA( ) method . . .
LOOSE for temp-table objects	• Matches temp-table columns by name. The data type and extent of the column in the XML Schema must match those for the matching column in the temp-table. Other field attributes in the XML Schema are ignored. • The XML Schema may be a subset or superset of the temp-table schema. Any columns that are in the XML Schema but not in the temp-table are ignored. Any columns that are in the temp-table, but not in the XML Schema, are ignored.

When the mode is . . .	The READ-XMLSCHEMA( ) method . . .
LOOSE for ProDataSet objects	• Matches temp-tables and columns by name. The data type and extent of the column in the XML Schema must match those for the matching column in the temp-table. Other field attributes in the XML Schema are ignored. • Data relationships are matched by parent buffer and child buffer names. For every data relationship in the XML Schema that matches a data-relation in the ProDataSet, the field mapping between the parent and child buffers must match. • The XML Schema may be a subset or superset of the ProDataSet schema. Any temp-tables, columns, or data-relations that are in the ProDataSet, but not in the XML Schema, are ignored. For a dynamic ProDataSet object, the method adds temp-tables and data-relations to the object when the temp-tables and data-relations are defined in the XML Schema, but are not members of the ProDataSet. Fields are not added to existing temp-tables. For a static ProDataSet object, any temp-tables or data-relations that are in the XML Schema, but not in the ProDataSet, are ignored.
STRICT for temp-table objects	• Matches temp-table columns by name. The data type and extent of the column in the XML Schema must match those for the matching column in the temp-table. Other field attributes in the XML Schema are ignored. • The XML Schema must define a field for every temp-table field, and there can be no extra fields in the XML Schema that are not in the temp-table schema.
STRICT for ProDataSet objects	• Matches temp-tables and columns by name. There must be a temp-table defined in the XML Schema for each table of the ProDataSet. There can be no tables defined in the XML Schema that are not in the ProDataSet schema. • There must be field defined in the XML Schema for each field in the temp-table, with the same data type and extent, and there can be no fields defined in the XML Schema that are not in the temp-table schema. • There must also be a data relationship defined in the XML Schema for every data-relation in the ProDataSet, and there can be no data relationships defined in the XML Schema that are not defined in the ProDataSet. The field mapping between the parent and child buffers must match.

If you specify the Unknown value (?), the method uses the default value of LOOSE.

If the XML Schema verification fails, the method generates an error message indicating the XML Schema element that caused the failure and returns FALSE.

If the ProDataSet or temp-table object does not have a schema (that is, the object is dynamic and in the CLEAR state), the AVM creates the schema from the XML Schema defined in the XML Schema document.

If the ProDataSet or temp-table object already has a schema (that is, the object is static, or the temp-tables are in the PREPARED state), the AVM verifies the XML Schema defined in the XML Schema document against the object's schema.

If a dynamic temp-table is not in the PREPARED or CLEAR state, the method generates an error and returns FALSE.

For more information about creating schema from XML Schema or verifying XML Schema, see *Use XML with ABL Applications*.

You cannot create schema for a temp-table buffer or a database buffer.

The following code example verifies the schema in a static ProDataSet object, in STRICT mode, using the schema defined in the specified XML Schema file:

```
DEFINE VARIABLE lRetOK                  AS LOGICAL   NO-UNDO.
DEFINE VARIABLE cSourceType             AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFile                   AS CHARACTER NO-UNDO.
DEFINE VARIABLE lOverrideDefaultMapping AS LOGICAL   NO-UNDO.
DEFINE VARIABLE cFieldTypeMapping       AS CHARACTER NO-UNDO.
DEFINE VARIABLE cVerifySchemaMode       AS CHARACTER NO-UNDO.

DEFINE TEMP-TABLE ttCustomer LIKE Customer NO-UNDO.
DEFINE TEMP-TABLE ttOrder    LIKE Order    NO-UNDO.
DEFINE TEMP-TABLE ttInvoive  LIKE Invoice  NO-UNDO.

DEFINE DATASET DSET FOR ttCustomer, ttOrder, ttInvoice
  DATA-RELATION CustOrd FOR ttCustomer,
    ttOrd RELATION-FIELDS(CustNum,CustNum) NESTED
  DATA-RELATION OrdInv FOR ttOrder,
    ttInv RELATION-FIELDS(OrderNum,OrderNum) NESTED.

ASSIGN
  cSourceType             = "file"
  cFile                   = "cust-ord-inv.xsd"
  lOverrideDefaultMapping = FALSE
  cFieldTypeMapping       = ?
  cVerifySchemaMode       = "strict".

lRetOK = DATASET DSET:READ-XMLSCHEMA (cSourceType, cFile,
  lOverrideDefaultMapping, cFieldTypeMapping,cVerifySchemaMode).
```

The following code example creates a dynamic temp-table object, creates the object's schema from the specified XML Schema file, and overrides the default data type mapping of one field:

```
DEFINE VARIABLE lRetOK                  AS LOGICAL   NO-UNDO.
DEFINE VARIABLE cSourceType             AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFile                   AS CHARACTER NO-UNDO.
DEFINE VARIABLE lOverrideDefaultMapping AS LOGICAL   NO-UNDO.
DEFINE VARIABLE cFieldTypeMapping       AS CHARACTER NO-UNDO.
DEFINE VARIABLE cVerifySchemaMode       AS CHARACTER NO-UNDO.
DEFINE VARIABLE hTable                  AS HANDLE    NO-UNDO.

CREATE TEMP-TABLE hTable.
ASSIGN
  cSourceType             = "file"
  cFile                   = "ttcust.xsd"
  lOverrideDefaultMapping = FALSE
  cFieldTypeMapping       = "address2,CLOB"
  cVerifySchemaMode       = ?.

lRetOK = hTable:READ-XMLSCHEMA (cSourceType, cFile, lOverrideDefaultMapping,
  cFieldTypeMapping,cVerifySchemaMode).
```

See also

IS-XML attribute, READ-XML( ) method, WEB-CONTEXT system handle, WRITE-XML( ) method, WRITE-XMLSCHEMA( ) method, SERIALIZE-ROW( ) method, X-document object handle, X-noderef object handle

RECID attribute

The unique internal identifier of the database record currently associated with the buffer.

Data type: RECID

Access: Read-only

Applies to: Buffer object handle

Note: ABL provides the RECID attribute (and the corresponding RECID function) for backward compatibility only. Progress Software Corporation recommends that in new code, you use the ROWID attribute.

Note: You cannot use a RECID to identify the partition of a record in a partitioned table. Using a stored RECID to retrieve a record in a partitioned table will not work unless the record is in the initial composite partition. Use ROWID instead.

RECORD-LENGTH attribute

The length, in bytes, of the record associated with a buffer.

Data type: INTEGER

Access: Read-only

Applies to: Buffer object handle

RECURSIVE attribute

Indicates if the data-relation is recursive.

Data type: LOGICAL

Access: Read-only

Applies to: Data-relation object handle

See also

ADD-PARENT-ID-RELATION( ) method, DEFINE DATASET statement, FILL( ) method

REFRESH( ) method (Handle)

Forces the AVM to refresh the display of the current rows in a browse.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
REFRESH ( )
```

If the AVM successfully refreshes the widget, the method returns the value TRUE.

REFRESHABLE attribute

Indicates whether the rows that appear in a browse are refreshed when an application opens or repositions a query.

Note: When an application opens a query or repositions it multiple times, and refreshes the viewport each time, the display might flash, which is distracting. You can suppress the refreshing, and so reduce the flashing, by setting REFRESHABLE to FALSE.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

If REFRESHABLE is FALSE, when an application opens or repositions a query, the viewport is not refreshed. REFRESHABLE's default value is TRUE.

REFRESH-AUDIT-POLICY( ) method

Notifies the specified audit-enabled database that its audit policy tables have changed, which causes the database to refresh its current run-time audit policy settings cache before performing any more database operations.

Return type: LOGICAL

Applies to: AUDIT-POLICY system handle

Syntax

```
REFRESH-AUDIT-POLICY ( integer-expression | logical-name | alias )
```

integer-expression

The sequence number of the connected database that needs to refresh its audit policy settings cache. For example, `REFRESH-AUDIT-POLICY(1)` notifies the first database, `REFRESH-AUDIT-POLICY(2)` notifies the second database, and so on. If you specify a sequence number that does not correspond to a connected database, the AVM generates a run-time error.

logical-name or *alias*

The logical name or alias of the connected database that needs to refresh its audit policy settings cache. These forms require a quoted character string or a character expression. If you specify a logical name or alias that does not correspond to a connected database, the AVM generates a run-time error.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

For information about audit-enabling a database, or creating and activating an audit policy for a database, see *Learn about Security and Auditing*.

REGISTER-DOMAIN( ) method

Registers a security domain in the ABL session domain registry. The AVM uses this registry to authenticate or validate (through a single sign-on (SSO) operation) the session identity represented by a client-principal object, as well as the connection identity for any OpenEdge database configured to use the session (application) registry.

Note: This method does not support the registration of domains to authenticate identity for multi-tenant database connections.

Caution: Use caution when registering domains in the session domain registry using the REGISTER-DOMAIN( ) method. You can introduce the risk for a security breach by allowing the registration of rogue domains between registering your own domains and locking the registry. Consider using the LOAD-DOMAINS( ) method, which loads all domain registry information directly from the local domain registry of a connected OpenEdge database (including multi-tenant databases).

Return type: LOGICAL

Applies to: SECURITY-POLICY system handle

Syntax

```
REGISTER-DOMAIN ( domain-name , access-code
  [ , domain-description [ , domain-type ] ] )
```

domain-name

A character expression that specifies the name of this security domain. For information on the restricted character set to use for this value, see the DOMAIN-NAME attribute entry.

access-code

A character expression that specifies the secret value to use when authenticating or validating a client-principal object that represents a user identity in this domain. The AVM converts this access code to UTF-8 before using it, which ensures a consistent value regardless of code page settings.

domain-description

An optional character expression that specifies a description for this domain.

domain-type

An optional character expression that specifies an application-defined authentication system for user authentication and single-sign-on (SSO) operations.

To authenticate or validate a user identity against a session domain registry built using REGISTER-DOMAIN( ), you must call the LOCK-REGISTRATION( ) method, which also locks and prevents any further registration of domains in the domain session registry.

If you do not register at least one domain in the session domain registry before calling LOCK-REGISTRATION( ), the LOCK-REGISTRATION( ) method returns TRUE. However, any attempt to seal a client-principal object against the session registry raises a run-time error.

If successful, this method returns TRUE. Otherwise, it returns FALSE. Note that you cannot call REGISTER-DOMAIN( ) successfully if you have already called the LOAD-DOMAINS( ) method or the LOCK-REGISTRATION( ) method.

The following code fragment illustrates how to use the REGISTER-DOMAIN( ) method:

```
DEFINE VARIABLE name AS CHARACTER NO-UNDO.
DEFINE VARIABLE key  AS CHARACTER NO-UNDO.

FOR EACH trusted-registrar:
  name = trusted-registrar.name.
  key = trusted-registrar.key.
  SECURITY-POLICY:REGISTER-DOMAIN(name, key).
END.

SECURITY-POLICY:LOCK-REGISTRATION.
```

See also

LOAD-DOMAINS( ) method, LOCK-REGISTRATION( ) method

REJECT-CHANGES( ) method

Rejects changes to the data in one temp-table or all temp-tables in a ProDataSet object.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle

Syntax

```
REJECT-CHANGES( )
```

When you reject changes on a ProDataSet object handle, the AVM uses the before-image table to back out changes from the after-image table, and empties the before-image table for each table in the ProDataSet.

When you reject changes for a Buffer object handle, the AVM uses the before-image table to back out changes from the after-image table, and empties the before-image table for that one table.

As the AVM rejects changes, it sets the BEFORE-ROWID attribute of every row in the after-image table to the Unknown value (?), and sets the ROW-STATE of every row in the after-image table to ROW-UNMODIFIED (0).

Note: When executing the method, there is no guarantee as to the contents of the buffers involved after the method finishes executing.

REJECT-ROW-CHANGES( ) method

Rejects changes to the data in one row of a ProDataSet temp-table.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
REJECT-ROW-CHANGES( )
```

When you reject changes for a temp-table row, the AVM uses the before-image table row to back out changes in the after-image table row. The AVM also sets the BEFORE-ROWID attribute of the row in the after-image table to the Unknown value (?), sets the ROW-STATE of the row in the after-image table to ROW-UNMODIFIED (0), and removes the before-image table row.

REJECTED attribute

Set this attribute to indicate whether a change to the data in a ProDataSet object, a temp-table buffer, or a temp-table row is rejected.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

The REJECTED attribute corresponds to the REJECTED function.

The MERGE-CHANGES( ) method and MERGE-ROW-CHANGES( ) method use this attribute to determine whether to reject a changed row during a merge operation.

This attribute is marshalled between the client and the application server.

Note: The REJECTED attribute is effective only if a before-table is NO-UNDO during a data source update. If the before-table is UNDO, the REJECTED attribute is lost.

RELATION-FIELDS attribute

Returns a comma-separated list of parent-field, child-field pairs describing the relationship between parent and child buffers as specified in the data-relation object definition.

Data type: CHARACTER

Access: Read-only

Applies to: Data-relation object handle

Returns a comma-delimited list as a character expression with the following syntax:

Syntax

```
"parent-field1,child-field1[,parent-fieldn,child-fieldn]..."
```

You can use the value of this attribute in writing code that uses or extends this list of join fields without having to parse the value of the WHERE-STRING attribute, which in the default case provides essentially the same information but not necessarily in an ideal form for analyzing the relation.

RELATIONS-ACTIVE attribute

Indicates whether all data-relation objects in a ProDataSet object are active or inactive. Set to TRUE to activate all data-relation objects. Set to FALSE to deactivate all data-relation objects. All data-relation objects in a ProDataSet object are active by default.

Alternatively, you can activate or deactivate an individual data-relation object in a ProDataSet object by setting the ACTIVE attribute on the Data-relation object handle.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: ProDataSet object handle

Deactivate all data-relations in a ProDataSet object when you want a FILL operation to load data into each ProDataSet member buffer using the individual buffer's query (instead of recursively loading parent and child buffers as defined by the data-relations). Likewise, you can reactivate all data-relations in a ProDataSet object after completing a FILL operation to use the data-relations for traversing the data after the data is loaded.

When the AVM encounters an inactive relation (or the last child buffer in the relation tree), during a FILL operation on a ProDataSet buffer object handle, the AVM does not fill the child buffers of that relation. When the AVM encounters an inactive relation during a FILL operation on a ProDataSet object handle, it treats the first child buffer of the inactive relation as a top-level table (including all records from its data source) and fills each child buffer based on the data relation's query. If you do not want the AVM to treat the first child buffer of the inactive relation as a top-level table, set the FILL-MODE of that buffer to NO-FILL. The AVM does not fill any of the child buffers.

If the AVM encounters an inactive relation while navigating a ProDataSet object, it does not prepare or open a dynamic query for the child table, even if there is a browse associated with the relation's query. If you want to access the child temp-table, you must do so through a separate query, a FOR EACH statement, or some other standard ABL construct in your application code.

When you reactivate data-relations, the AVM does not automatically resynchronize the hierarchy of queries on buffers below the newly active relation. If you want to resynchronize the related buffers, use the SYNCHRONIZE( ) method on the parent buffer.

REMOTE attribute

Indicates whether the specified procedure is running at the top level of an application server session as the result of a remote procedure call from a client application, or whether the current ABL session is an application server session. Returns FALSE for a Web service procedure.

Data type: LOGICAL

Access: Read-only

Applies to: Procedure object handle, SESSION system handle

For any procedure handle, REMOTE is TRUE if:

- The specified procedure is running locally at the top level of the current session.
- The current session is an application server session.
- The procedure is running directly as the result of a remote procedure call from a client application.

Otherwise, REMOTE is FALSE. Thus, if the procedure handle is a proxy handle (PROXY attribute set to TRUE) or the specified procedure is running as the direct result of a call from any other procedure running in the current session context, REMOTE is FALSE.

For the SESSION handle, REMOTE is TRUE if the session runs in the context of an application server, and FALSE if the session runs in the context of an ABL client.

REMOTE-HOST attribute

Indicates the IP (Internet Protocol) address of the machine with which the socket object is communicating.

Data type: CHARACTER

Access: Read-only

Applies to: Socket object handle

When a server and client successfully establish a connection, both the server and client have a socket object that identifies this connection. On the client, this attribute returns the IP address of the server, and on the server, this attribute returns the IP address of the client. If the CONNECT( ) method fails or has not been called, this attribute returns the Unknown value (?).

See also

DB-REMOTE-HOST function

REMOTE-PORT attribute

Indicates the port number of the socket.

Data type: INTEGER

Access: Read-only

Applies to: Socket object handle

When a server and client successfully establish a connection, both the server and client have a socket object that identifies this connection. On the client, this attribute returns the port number used on the server machine for this socket connection. On the server, this attribute returns the port number used on the client machine for this socket connection. If the CONNECT failed, this attribute returns the Unknown value (?).

REMOVE-ATTRIBUTE( ) method

Removes the specified attribute of an element. If the removed attribute has a default value (specified by the document's DTD) it is set to its default value.

Return type: LOGICAL

Applies to: SAX-attributes object handle, X-noderef object handle

Syntax

```
REMOVE-ATTRIBUTE ( { attribute-name | index } [ , namespaceURI ] )
```

attribute-name

A CHARACTER expression evaluating to the fully qualified or unqualified name of the attribute.

index

For SAX-attributes object handle only. An integer expression evaluating to the position of the attribute in the attribute list. The first attribute has an index of 1. When using *index*, *namespaceURI* must be the Unknown value (?), an empty string, or not supplied. Otherwise, an invalid argument error will be raised.

namespaceURI

A CHARACTER expression evaluating to:

- The URI of the attribute
- A zero-length string
- The Unknown value (?) if the attribute doesn't contain a namespace

Call this method to remove an attribute from a SAX-attributes object or an X-noderef object. For example, if an XML element obtained from the SAX-reader object has a larger set of attributes than that required by the document being written by the SAX-writer, use this method to find and remove the extra attributes.

The index values of attributes left in the list is then updated to reflect the removed attribute. The SAX-attributes NUM-ITEMS attribute value will also be decremented to reflect the removed attribute.

If the method fails to find a match, because the name is not found or the index position is invalid, then the method will not remove an attribute and return FALSE.

If using an *attribute-name*, the method removes the first match it finds in the list.

The following are examples for a SAX-attributes object:

```
DEFINE VARIABLE hSAX-attributes AS HANDLE NO-UNDO.

CREATE SAX-ATTRIBUTES hSAX-attributes.

hSAX-attributes:INSERT-ATTRIBUTE( "language", "EN" ). /* index = 1 */
hSAX-attributes:INSERT-ATTRIBUTE( "year", "2006" ).   /* index = 2 */

hSAX-attributes:REMOVE-ATTRIBUTE( "language" ).
hSAX-attributes:REMOVE-ATTRIBUTE( 1 ).                /* removes "year" */
```

The following X-noderef object example removes the attribute "Id", or resets the attribute "Id" to its default value, for the XML node represented by hNoderef:

```
hNoderef:REMOVE-ATTRIBUTE("Id").
```

REMOVE-CHILD( ) method

Unlinks the node and its sub-tree from the XML document. The XML object is not deleted, only disconnected from the structure.

Return type: LOGICAL

Applies to: X-document object handle, X-noderef object handle

Syntax

```
REMOVE-CHILD ( x-node-handle )
```

x-node-handle

The handle that represents the node to remove from the tree.

The following code fragment gets a reference to the fourth node on the document root, and removes it. hNoderef is still available for use after the remove, but is unlinked from hRoot:

```
CREATE X-NODEREF hNoderef.
. . .
hRoot:GET-CHILD(hNoderef,4).
hRoot:REMOVE-CHILD(hNodeRef).
. . .
```

REMOVE-EVENTS-PROCEDURE( ) method
(Windows only; Graphical interfaces only)

Removes an external procedure from the list that the AVM searches for event procedures to handle an ActiveX control event.

Return type: LOGICAL

Applies to: CONTROL-FRAME widget

Syntax

```
REMOVE-EVENTS-PROCEDURE ( procedure-handle )
```

procedure-handle

A handle to a persistent procedure or an otherwise active procedure on the call stack.

After removing the handle to an external procedure containing an event handler, the equivalent event procedure found in the next procedure in the search list handles the event. For information on this search list, see the ADD-EVENTS-PROCEDURE( ) method reference entry.

If the method succeeds in removing the procedure file from the list, it returns TRUE. Otherwise, it returns FALSE.

REMOVE-SUPER-PROCEDURE( ) method

Dissociates a super procedure file from a procedure file or from the current ABL session. Returns FALSE for a Web service procedure.

Note: Dissociating a super procedure file from the current ABL session does not automatically dissociate the super procedure file from procedure files within the session.

Return type: LOGICAL

Applies to: Procedure object handle, SESSION system handle

Syntax

```
REMOVE-SUPER-PROCEDURE ( super-proc-hdl  )
```

super-proc-hdl

A handle to the super procedure.

Note: If *super-proc-hdl* is not a valid procedure handle or is not currently a super procedure of the local procedure or of the current ABL session, the AVM does not report a run-time error.

REMOVE-SUPER-PROCEDURE returns FALSE if *super-proc-hdl* is not a valid handle. Otherwise, it returns TRUE.

The following code fragment dissociates a super procedure from the current procedure:

```
THIS-PROCEDURE:REMOVE-SUPER-PROCEDURE(my-super-proc-hdl).
```

The following code fragment dissociates a super procedure from a procedure file other than the current procedure:

```
local-proc-hdl:REMOVE-SUPER-PROCEDURE(my-super-proc-hdl).
```

The following code fragment dissociates a super procedure from the current ABL session:

```
SESSION:REMOVE-SUPER-PROCEDURE(my-super-proc-hdl).
```

REPLACE( ) method

Replaces an item in a combo box, radio set, or selection list. Replaces an existing text string in an editor with a new text string.

Return type: LOGICAL

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, RADIO-SET widget, SELECTION-LIST widget

Syntax (combo-box, selection-list, or browse column)

This is the syntax for combo-box, selection-list or browse column.

```
REPLACE ( { new-item-list | new-label , new-value }
  , { list-item | list-index } )
```

new-item-list

A character-string expression that specifies a single item or a delimiter-separated list of items to add to the widget.

new-label

A character-string expression that specifies the label of a label-value pair to add to the widget.

new-value

The new value assigned when a user selects the label.

Note: Use *new-item-list* when the widget's entries consist of single items. Use *new-label* and *new-value* when the widget's entries consist of label-value pairs.

list-item

A character-string expression that specifies a single value in the widget.

list-index

An integer expression that specifies the ordinal position of an existing entry in the combo box list or selection list.

For combo boxes and selection lists, REPLACE replaces *list-item* with either *new-label-list* or the label-value pair represented by *new-label* and *new-value*. If *list-item* is currently selected, the new item is not selected when it appears in the list. If the method is successful, it returns TRUE.

For browses, this method applies only to combo-box browse columns.

Syntax (editor)

This is the syntax for an editor:

```
REPLACE ( old-string , new-string , flag )
```

old-string

A character-string expression to be replaced.

Note: You cannot specify wildcard characters in the search string.

new-string

A character-string expression to replace *old-string*.

flag

An integer expression that specifies the type of search to be performed.

For editors, REPLACE searches from the current text cursor position for an occurrence of *old-string* and replaces it with *new-string*. If the replace operation is successful, the method returns TRUE. The *flag* value determines the type of search and replace to perform. The following table lists the flag values that correspond to each search and replace type.

Table 101: REPLACE flag values

Type of search	Flag value
FIND-NEXT-OCCURRENCE	1
FIND-PREV-OCCURRENCE	2
FIND-CASE-SENSITIVE	4
FIND-GLOBAL	8
FIND-WRAP-AROUND	16

For a single operation, you cannot specify both FIND-NEXT-OCCURRENCE and FIND-PREV-OCCURRENCE, nor can you specify both FIND-WRAP-AROUND and FIND-GLOBAL. All other combinations of flags are valid. For example, you can specify a combination of FIND-NEXT-OCCURRENCE + FIND-GLOBAL + FIND-CASE-SENSITIVE. The default is FIND-NEXT_OCCURRENCE to search to the end of the editor string.

Syntax (radio-set)

```
REPLACE ( new-label , new-value , old-label )
```

new-label

A character-string expression that specifies the new item label.

new-value

The new value assigned when a user selects the item.

old-label

A character-string expression that specifies the label of the item to be replaced.

The REPLACE( ) method for radio sets replaces the label, the value, or both the label and value of the specified radio item. To retain the existing label or value, substitute an empty string.

REPLACE(*new-label* , *new-value* , *old-label*) replaces the specified radio item with a new item, consisting of both a new label and a new value.

REPLACE(*new-label* , *""* , *old-label*) replaces only the label of the specified radio item, retaining the value.

REPLACE(*""* , *new-value* , *old-label*) replaces only the value of the specified radio item, retaining the label.

If the new label is longer than the existing radio set size can accommodate, the radio set appearance changes depending on setting of the AUTO-RESIZE attribute. If AUTO-RESIZE is TRUE, the radio set expands to accommodate the label. If AUTO-RESIZE is FALSE, the new label is clipped to fit the current size. However, note that the label is clipped only on the display. The new radio set item is identified by the full label regardless of its length.

If the replace operation is successful, the method returns TRUE.

REPLACE-CHILD( ) method

Replace an old XML node with a new node. The old XML node is not deleted, only disconnected from the structure. If the new XML node is already in the tree, it is first disconnected.

Return type: LOGICAL

Applies to: X-document object handle, X-noderef object handle

Syntax

```
REPLACE-CHILD ( new-handle , old-handle )
```

new-handle

The handle that represents the node to insert in the tree.

old-handle

The handle that represents the node to remove from the tree.

The following code fragment gets a reference to the fourth XML node on the document root, and removes it. hNoderef is still available for use after the remove, but is unlinked from hRoot. We then replace the root's second child with this fourth child:

```
CREATE X-NODEREF hNoderef.
CREATE X-NODEREF hChild.
...
hRoot:GET-CHILD(hNoderef,4).
hRoot:REMOVE-CHILD(hNodeRef).
hRoot:GET-CHILD(hChild,2).
hRoot:REPLACE-CHILD(hNodeRef, hChild).
```

REPLACE-SELECTION-TEXT( ) method

Replaces the currently selected text in an editor widget with the new text.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
REPLACE-SELECTION-TEXT ( new-text )
```

new-text

A character-string expression that specifies the new text to replace the currently selected text.

To determine what text is currently selected, query the SELECTION-TEXT attribute. If the replace operation is successful, the method returns TRUE.

REPOSITION attribute

The reposition mode of a data-relation object. If TRUE, the relation mode is REPOSITION. If FALSE, the relation mode is SELECTION. The default value is FALSE.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Data-relation object handle

REPOSITION-BACKWARD( ) method

Moves a query object's result list pointer back a particular number of rows.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
REPOSITION-BACKWARD ( n )
```

n

An integer expression representing the number of rows.

REPOSITION-BACKWARD( ) always places the cursor between rows. For example:

- If the cursor in on a row—say, row 5—REPOSITION-BACKWARD(1) moves the cursor to row 4, then to half way between rows 4 and 5. From this position, GET-PREV( ) moves the cursor to row 4, while GET-NEXT( ) moves the cursor to row 5.
- If the cursor is between two rows—say, between rows 5 and 6— REPOSITION-BACKWARD(1) moves the cursor to half way between rows 4 and 5. From this position, GET-PREV( ) moves the cursor to row 4, while GET-NEXT( ) moves the cursor to row 5.

Note: The REPOSITION-BACKWARD method corresponds to the REPOSITION statement BACKWARDS option.

REPOSITION-FORWARD( ) method

Moves a query object's result list pointer forward a particular number of rows.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
REPOSITION-FORWARD ( n )
```

n

An integer expression representing the number of rows.

REPOSITION-FORWARD( ) always places the cursor between rows. For example:

- If the cursor in on a row—say, row 5—REPOSITION-FORWARD(1) moves the cursor to row 6, then to half way between rows 6 and 7. From this position, GET-PREV( ) moves the cursor to row 6, while GET-NEXT( ) moves the cursor to row 7.

- If the cursor is between two rows—say, between rows 5 and 6— REPOSITION-FORWARD(1) moves the cursor to half way between rows 6 and 7. From this position, GET-PREV( ) moves the cursor to row 6, while GET-NEXT( ) moves the cursor to row 7.

Note: The REPOSITION-FORWARD method corresponds to the REPOSITION statement with the FORWARDS option.

See also

REPOSITION statement, REPOSITION-TO-ROW( ) method, REPOSITION-TO-ROWID( ) method

REPOSITION-TO-ROW( ) method

Moves a query object's result list pointer to the row corresponding to the specified sequence number.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
REPOSITION-TO-ROW ( n )
```

n

An integer expression representing the sequence number.

Note: The REPOSITION-TO-ROW method corresponds to the REPOSITION statement with the TO ROW option.

See also

REPOSITION statement, REPOSITION-FORWARD( ) method, REPOSITION-TO-ROWID( ) method

REPOSITION-TO-ROWID( ) method

Moves a query object's result list pointer to the row corresponding to the specified ROWID or ROWIDs.

To reposition to a particular row when the query is a join, supply the ROWID values of the buffers that correspond to the desired row.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
REPOSITION-TO-ROWID (
  {    rowid1 [ , rowid2 ] [ , tenant-expression ]
     | rowid-array
  }
```

rowid1 **[** *, rowid2* **] . . .**

Expressions of type ROWID representing the ROWID of the first buffer, the ROWID of the second buffer, and so on, in the join. The maximum number of expressions is 18. If an expression contains the Unknown value (?), the AVM evaluates but ignores subsequent expressions.

rowid-array

An array of 18 or fewer elements, where each element is of type ROWID and represents the rowid of a buffer. If an element contains the Unknown value (?), the AVM evaluates but ignores subsequent elements.

tenant-expression

This parameter is useful only for a multi-tenant database, and primarily one with a connection identity that has super tenant access. If the user has a super-tenant connection identity and you **do not** specify this parameter, the query repositions to data owned by the effective tenant. If you **do** specify this parameter, the query repositions to data owned by the regular tenant identified by *tenant-expression*.

If the user has a regular-tenant connection identity, and you specify this parameter, *tenant-expression* must match the tenancy of the connection identity. Otherwise, the statement raises ERROR.

If *tenant-expression* evaluates to an integer, the value must be a valid tenant ID for a regular tenant or zero (0) for the default tenant. If *tenant-expression* evaluates to a character string, the value must be a valid tenant name for a regular or "Default" for the default tenant. Otherwise, the statement raises ERROR.

If you pass *tenant-expression* and any of the ROWID values belong to a table that is not multi-tenant enabled, ABL raises a run-time error.

Notes

- The REPOSITION-TO-ROWID( ) method corresponds to the REPOSITION statement with the TO ROWID option.
- When you pass *tenant-expression*, the AVM looks up the tenant identity in the database with a share lock. The AVM waits 60 seconds to get the share lock and raises ERROR if it fails to obtain the share lock in that amount of time. The AVM releases the share lock immediately after successfully fetching the row. This share lock is released even if the statement is called while in the scope of a transaction.

See also

REPOSITION statement, REPOSITION-FORWARD( ) method, REPOSITION-TO-ROW( ) method

REQUEST-INFO attribute

An object reference to a `Progress.Lang.OERequestInfo` class that provides information about a client request sent from this ABL session to an application server.

Data type: Progress.Lang.OERequestInfo class

Access: Read-only

Applies to: Server object handle, Asynchronous request object handle

The following code fragment retrieves the default client context identifier (CCID) that the AVM generates when you create a server object handle:

```
DEFINE VARIABLE hAppSrv  AS HANDLE NO-UNDO.
DEFINE VARIABLE ccid     AS CHARACTER NO-UNDO.

CREATE SERVER hAppSrv.
ccid = hAppSrv:REQUEST-INFO:ClientContextId.
```

You can also access this attribute using the SELF system handle inside an ABL event procedure or method that runs when an associated asynchronous remote procedure completes.

The following code fragment defines an event procedure (`AsynReqIsDone`) that you might use to access this attribute in response to an asynchronous remote procedure call:

```
DEFINE VARIABLE ccid AS CHARACTER NO-UNDO.

/* Assume that this procedure has been specified as the event procedure */
/* for an asynchronous remote procedure call */

PROCEDURE AsyncRqIsDone:

  ccid = SELF:REQUEST-INFO:ClientContextId.
  DELETE OBJECT SELF.

END.
```

This event procedure accesses the CCID that was sent to the application server along with the asynchronous request and assigns the value to the `ccid` variable defined in the enclosing external procedure.

The bold code in the following fragment shows setting a client principal in a server's `OERequestInfo` instance:

```
DEFINE VARIABLE hAppSrv      AS HANDLE.
DEFINE VARIABLE hCPIn        AS HANDLE.
DEFINE VARIABLE hCPOut       AS HANDLE.
DEFINE VARIABLE ccid         AS CHARACTER.
DEFINE VARIABLE bOK          AS LOGICAL.
DEFINE VARIABLE loginState   AS CHARACTER.

CREATE SERVER hAppSrv.
ccid = hAppSrv:REQUEST-INFO:clientContextID.

CREATE CLIENT-PRINCIPAL hCPIn.
hCPIn:INITIALIZE("userName@domainName", ccid, ?, "password").

hAppSrv:REQUEST-INFO:SetClientPrincipal(hCPIn).

bOK = hAppSrv:CONNECT("-URL AppServerDC://asHost:asPort/svc").

hCPOut = hAppSrv:RESPONSE-INFO:GetClientPrincipal().
IF VALID-HANDLE(hCPOut)
THEN loginState = hCPOut:LOGIN-STATE.
ELSE loginState = ?.

. . .

IF VALID-HANDLE(hCPOut)
   THEN DELETE OBJECT hCPOut.
```

See also

GetClientPrincipal( ) method (Progress.Lang.OERequestInfo), CURRENT-REQUEST-INFO attribute, CURRENT-RESPONSE-INFO attribute, LOCAL-VERSION-INFO attribute, RESPONSE-INFO attribute, SetClientPrincipal( ) method

RESET( ) method

Closes the open stream and resets the SAX-writer object to its default values.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
RESET( )
```

Use this method to reuse a SAX-writer object for multiple documents or to cancel a write.

The object attributes and the output destination remain unchanged. The WRITE-STATUS attribute is set to SAX-WRITE-IDLE.

You can call this method at any time. If the WRITE-STATUS attribute is either SAX-WRITE-IDLE or SAX-WRITE-COMPLETE, the method call has no effect. Otherwise, the method call closes the document and stream, and aborts the write.

See also

WRITE-STATUS attribute

RESIZABLE attribute
(Graphical interfaces only)

Indicates if the user can resize a widget at run time.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

If you set RESIZABLE to TRUE, the user can resize the widget. For the RESIZABLE attribute to take effect, you must also set the SELECTABLE attribute to TRUE.

Note: Setting the RESIZABLE attribute to TRUE enables direct manipulation events for the widget. These events take precedence over all other events. This effectively prevents data entry using the widget until all direct manipulation events are disabled (that is, until MOVABLE, RESIZABLE, and SELECTABLE are all FALSE).

RESIZE attribute
(Graphical interfaces only)

Indicates if the user can resize a window at run time.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WINDOW widget

If the RESIZE attribute is TRUE, users can resize the window.

You can set this attribute only before the window is realized.

RESPONSE-INFO attribute

An object reference to a `Progress.Lang.OERequestInfo` class that provides information returned from an application server about its response to an application server client request.

Data type: Progress.Lang.OERequestInfo class

Access: Readable/Writeable

Applies to: Server object handle, Asynchronous request object handle

The following code fragment retrieves the client context identifier (CCID) that is returned from a call to the remote procedure, `remote.p`:

```
DEFINE VARIABLE hAppSrv  AS HANDLE NO-UNDO.
DEFINE VARIABLE ccid     AS CHARACTER NO-UNDO.
DEFINE VARIABLE bOK      AS LOGICAL.

CREATE SERVER hAppSrv.
bOK = hAppSrv:CONNECT("-URL AppServerDC://asHost:asPort/svc").
RUN remote.p ON SERVER hAppSrv.

ccid = hAppSrv:RESPONSE-INFO:ClientContextId.
```

You can also access this attribute from inside of an ABL event procedure or method that runs when an associated asynchronous remote procedure call completes. This attribute is read-only on the asynchronous request object handle that you access within the event procedure or method using the SELF system handle.

The following code fragment defines an event procedure (`AsynReqIsDone`) that you might use to access this attribute in response to an asynchronous remote procedure call:

```
DEFINE VARIABLE ccid AS CHARACTER NO-UNDO.

/* Assume the following procedure has been specified as the  */
/* event procedure for an asynchronous remote procedure call */

PROCEDURE AsynRqIsDone:

  ccid = SELF:RESPONSE-INFO:ClientContextId.
  DELETE OBJECT SELF.

END.
```

This event procedure accesses the CCID that is returned from the application server and assigns the value to the `ccid` variable defined in the enclosing external procedure.

The bold code in the following fragment shows getting a client principal from a server's server's `OERequestInfo` instance:

```
DEFINE VARIABLE hAppSrv      as HANDLE.
DEFINE VARIABLE hCPIn        as HANDLE.
DEFINE VARIABLE hCPOut       as HANDLE.
DEFINE VARIABLE ccid         as CHARACTER.
DEFINE VARIABLE bOK          as LOGICAL.
DEFINE VARIABLE loginState   as CHARACTER.

CREATE SERVER hAppSrv.
ccid = hAppSrv:REQUEST-INFO:clientContextID.

CREATE CLIENT-PRINCIPAL hCPIn.
hCPIn:INITIALIZE("userName@domainName", ccid, ?, "password").

hAppSrv:REQUEST-INFO:SetClientPrincipal(hCPIn).

bOK = hAppSrv:CONNECT("-URL AppServerDC://asHost:asPort/svc").

hCPOut = hAppSrv:RESPONSE-INFO:GetClientPrincipal().
IF VALID-HANDLE(hCPOut)
THEN loginState = hCPOut:LOGIN-STATE.
ELSE loginState = ?.

. . .

IF VALID-HANDLE(hCPOut)
   THEN DELETE OBJECT hCPOut.
```

See also

GetClientPrincipal( ) method (Progress.Lang.OERequestInfo), CURRENT-REQUEST-INFO attribute, CURRENT-RESPONSE-INFO attribute, LOCAL-VERSION-INFO attribute, REQUEST-INFO attribute, SetClientPrincipal( ) method

RESTART-ROW attribute

Allows the FILL query to be repositioned to an absolute row number. Similar to RESTART-ROWID, this attribute facilitates batching during a FILL on a ProDataSet temp-table.

Data type: INT64

Access: Readable/Writeable

Applies to: Data-source object handle

Setting this attribute is helpful when you are paging back and forth in a table and want to retrieve, for example, page 3. The BATCH-SIZE attribute is used to calculate the row where a given page begins, and RESTART-ROW is set to that value just before the FILL. This causes the FILL query to be repositioned to the correct "page". For example, to get page 3 you set RESTART-ROW to (3-1) * BATCH-SIZE to fill the third group of records.

See also

BATCH-SIZE attribute, RESTART-ROWID attribute

RESTART-ROWID attribute

Specifies the ROWID of the data source row at which a FILL operation will start. Set this attribute before each FILL operation in a series of FILL operations to retrieve data source rows in batches. You typically set this attribute by assigning the value of the NEXT-ROWID attribute, which is set by the AVM after each FILL operation.

Note: The NEXT-ROWID attribute is not marshalled between the client and the application server. You are responsible for retrieving, storing, and transporting this attribute value between the client and the application server.

Data type: ROWID

Access: Readable/Writeable

Applies to: Data-source object handle

Syntax

```
RESTART-ROWID ( buffer-sequence-number | buffer-name )
```

buffer-sequence-number

An integer that represents the sequence number of a buffer in the list of buffers for the data-source object. Specify *buffer-sequence-number* to identify a buffer in the data-source object when the data-source object is defined against more than one database table buffer. The default is the first (or only) buffer in the data-source object.

Note: Sequence numbers for buffers in a data-source object start at one, where one represents the top level and subsequent numbers represent lower levels of join, if any.

buffer-name

A CHARACTER expression that evaluates to the name of a buffer in the list of buffers for the data-source object.

If an invalid buffer is specified, this attribute returns the Unknown value (?).

When specified, the next FILL operation using this data source opens its associated query and tries to reposition the query to the given ROWID. If the reposition is successful, the AVM proceeds with the FILL operation. If the reposition is not successful, the AVM sets the RESTART-ROWID attribute to the Unknown value (?) and proceeds with the FILL operation from the top of the query at the current level (that is, without repositioning the query).

This attribute is not marshalled between the client and the application server.

See also

BATCH-SIZE attribute, FILL( ) method, NEXT-ROWID attribute, RESTART-ROW attribute

RETAIN-SHAPE attribute

Indicates that the image should retain its aspect ratio (expand or contract equally in both dimensions).

Data type: LOGICAL

Access: Readable/Writeable

Applies to: IMAGE widget

Setting RETAIN-SHAPE to TRUE may leave some uncovered space at the bottom or right of the image widget. RETAIN-SHAPE is ignored if STRETCH-TO-FIT is equal to FALSE or if an icon is displayed on the image widget.

RETURN-INSERTED attribute
(Windows only)

How an editor widget behaves when a RETURN event occurs.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: EDITOR widget

If the RETURN-INSERTED attribute is TRUE, a RETURN event inserts a hard return at the cursor position, breaking the current line. Otherwise, if the editor is in a dialog box with a default button, a RETURN event chooses the default button for the dialog box. If the editor is not in a dialog box with a default button, a RETURN event inserts a hard return for any value of RETURN-INSERTED. The default value is FALSE.

You can set this attribute only before the editor widget is realized.

RETURN-VALUE attribute

The value returned from a user-defined function, attribute, or method, or a Windows DLL routine, or a UNIX shared library routine.

Data type: The value of the RETURN-VALUE-DATA-TYPE attribute.

Access: Read-only

Applies to: Call object handle

The default is the Unknown value (?).

If you set the RETURN-VALUE-DATA-TYPE attribute to a particular data type before executing INVOKE( ), the value returned is converted to that data type.

If RETURN-VALUE-DATA-TYPE is set to its default value or to the expected data type, when the dynamic invoke returns, the value returned by the user-defined function or attribute is not converted.

When invoking a Windows DLL routine or UNIX shared library routine, RETURN-VALUE-DLL-TYPE must be set before invoking the routine. If you do not, then any value returned by the DLL routine or the UNIX shared library routine will be ignored and RETURN-VALUE will be set to the Unknown value (?).

See also

RETURN-VALUE-DATA-TYPE attribute, RETURN-VALUE-DLL-TYPE attribute

RETURN-VALUE-DATA-TYPE attribute

When you dynamically invoke a user-defined function, get an attribute, or run a method, RETURN-VALUE-DATA-TYPE lets you do one of the following to the return value:

- Detect its data type
- Cause it to be converted to a different data type

If you set RETURN-VALUE-DATA-TYPE to a data type other than the expected data type before executing the dynamic invoke, the value returned is automatically converted to that data type. Otherwise, when the dynamic invoke returns, RETURN-VALUE-DATA-TYPE indicates the data type of the value returned.

Data type: CHARACTER

Access: Readable/Writable

Applies to: Call object handle

Syntax

```
RETURN-VALUE-DATA-TYPE [ = datatype ]
```

datatype

A character expression equal to one of the following:

- "CHARACTER"
- "DATE"
- "DATETIME"
- "DATETIME-TZ"
- "DECIMAL"
- "HANDLE"
- "INT64"
- "INTEGER"
- "LOGICAL"
- "RAW"
- Unknown value (?)

The default is the Unknown value (?).

RETURN-VALUE-DLL-TYPE attribute

Specifies the expected data type returned from a Windows Dynamic Link Library (DLL) routine.

Data type: CHARACTER

Access: Readable/Writable

Applies to: Call object handle

The RETURN-VALUE-DLL-TYPE attribute must be assigned before invoking the call object's INVOKE( ) method. After invocation, RETURN-VALUE contains an equivalent ABL data type which matches RETURN-VALUE-DLL-TYPE as shown in the following table.

Table 102: Mapping RETURN-VALUE-DLL-TYPE to RETURN-VALUE

RETURN-VALUE-DLL-TYPE	RETURN-VALUE
BYTE	INTEGER
CHARACTER	CHARACTER
DOUBLE	DECIMAL
FLOAT	DECIMAL
INT64	INT64
LONG	INTEGER
MEMPTR	MEMPTR
SHORT	INTEGER
UNSIGNED-LONG	INT64
UNSIGNED-SHORT	INTEGER

For example, if a DLL function is expected to return a LONG value, then RETURN-VALUE-DLL-TYPE is set to `"LONG"` before executing the INVOKE( ) method. After the invocation, RETURN-VALUE-DATA-TYPE has a value of `"INTEGER"` and RETURN-VALUE contains an INTEGER value.

If RETURN-VALUE-DLL-TYPE is not set, the DLL routine's return value is ignored, and RETURN-VALUE is set to the Unknown value (?).

See also

INVOKE( ) method (Handle), RETURN-VALUE attribute

ROLE attribute
(SOAP 1.2 only)

Returns the value of the role attribute for the SOAP-header-entryref object as a URI.

Data type: CHARACTER

Access: Read-only

Applies to: SOAP-header-entryref object handle

If the SOAP-header-entryref object does not contain a ROLE attribute, this attribute returns the empty string.

Note: In SOAP Version 1.2, the ROLE attribute replaces the SOAP Version 1.1 ACTOR attribute. If the ABL client is connected via SOAP Version 1.1, the ROLE attribute returns the empty string.

ROLES attribute

Returns a comma-separated list of domain roles for the user identity associated with the client-principal object. This list cannot contain embedded spaces. If not specified, the AVM returns a zero-length character string.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

Once the client-principal object is sealed, this attribute is read-only, and attempting to write to it raises a run-time error.

Note: You can use this attribute with the CAN-DO function, for example, to identify application functions accessible to a user both according to their user ID and according to their role.

See also

CAN-DO function

ROUNDED attribute
(Graphical interfaces only)

Indicates whether the rectangle appears with rounded corners.

Data type: LOGICAL

Access: Readable/Writable

Applies to: RECTANGLE widget

When TRUE, the rectangle appears with rounded corners (whether the display is set to the Windows XP Theme or the Windows Classic Theme). When FALSE, the rectangle appears with square corners.

If you set this attribute after the widget is realized, its affect is immediate (if the widget is visible).

See also

GROUP-BOX attribute

ROW attribute

The row position of the top edge of the widget relative to the top edge of the current iteration of a parent widget or the display. Specifies the row position of the mouse cursor for the last mouse event relative to the top edge of the display.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: BROWSE widget (browse and cell), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget, LAST-EVENT system handle

This attribute is read-only for browse columns, field groups, and the LAST-EVENT handle.

For all widgets except windows, the ROW attribute specifies the location, in character units, of the top edge of the widget relative to the top edge of its parent widget. In windows, the location is relative to the top edge of the display.

For a browse column, the ROW attribute returns the Unknown value (?) if the column is hidden.

If the parent is a down frame with multiple occurrences, the ROW attribute regards the parent as the current occurrence.

For control-frames, the ROW attribute maps to the Top property of the control-frame COM object (ActiveX control container).

For the LAST-EVENT handle, the ROW attribute specifies the row location, in character units, of the last mouse event relative to the top edge of the current frame.

This attribute is functionally equivalent to the Y attribute.

ROW-HEIGHT-CHARS attribute
(Graphical interfaces only)

Sets the row height, in characters, of a browse.

Data type: DECIMAL

Access: Readable/Writable

Applies to: BROWSE widget

All rows of a browse have the same height.

The AVM sets the ROW-HEIGHT-CHARS attribute of a browse that contains a combo-box column to the height of the combo-box, by default. You can override this setting after the browse is created.

ROW-HEIGHT-PIXELS attribute
(Graphical interfaces only)

Sets the row height, in pixels, of a browse.

Data type: INTEGER

Access: Readable/Writable

Applies to: BROWSE widget

All rows of a browse have the same height.

The AVM sets the ROW-HEIGHT-PIXELS attribute of a browse that contains a combo-box column to the height of the combo-box, by default. You can override this setting after the browse is created.

ROW-STATE attribute

The current change state of the ProDataSet temp-table row associated with the buffer object handle.

Data type: INTEGER

Access: Read-only

Applies to: Buffer object handle

Note: The ROW-STATE attribute corresponds to the ROW-STATE function.

When the TRACKING-CHANGES attribute is set to TRUE for a ProDataSet temp-table, the AVM tracks changes to the data in that temp-table using a before-image temp-table that contains the original version of each row. You can think of the temp-table itself as the after-image because it contains the latest version of each row.

Every row in the after-image table that has been modified or created corresponds to a row in the before-image table. Deleted rows do not appear in the after-image table, because it reflects the current state of the data. Every row in the before-image table has a non-zero ROW-STATE, because every row is the before-image of a deleted, created, or modified row in the after-image table. Unchanged rows do not appear in the before-image table.

You can use the ROW-STATE attribute on each row in either the after-image table or the before-image table to determine whether a row has changed and how it has changed.

The possible values can be expressed as compiler constants. The following table lists these values.

Table 103: Row state values

Compiler constant	Value	Description
ROW-UNMODIFIED	0	The row was not modified.
ROW-DELETED	1	The row was deleted.
ROW-MODIFIED	2	The row was modified.
ROW-CREATED	3	The row was created.

ROWID attribute

The unique internal identifier of the database record currently associated with the buffer.

Data type: ROWID

Access: Read-only

Applies to: Buffer object handle

Note: The ROWID attribute corresponds to the ROWID function. The ROWID of a record in a partitioned table identifies both the partition and the address of the record in the partition. If a partition field is updated in the record and, as a result, the record is moved to a different partition, the record's ROWID will change.

See also

ROWID function

ROW-MARKERS attribute

Indicates whether a browse uses row markers.

Data type: LOGICAL

Access: Read-only

Applies to: BROWSE widget

If this attribute is set to TRUE, the browse has row markers.

ROW-RESIZABLE attribute
(Graphical interfaces only)

Indicates whether you can change a browse's row height.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget

If ROW-RESIZABLE is TRUE, the AVM sensitizes the browse to the START-ROW-RESIZE and END-ROW-RESIZE events, which lets the user change the row height at any row juncture.

Alternately, if you add the ResizeinCell option to the [Startup] section of the `progress.ini` file and set it to NO, you limit the user's ability to change the row height to only at the row marker separator. If a row does not have a row marker, the user cannot change the height of that row.

If the ResizeinCell option is set to YES, you get the default behavior where the user can change the row height at any row juncture.

Note: In a browse, all rows have the same height.

SAVE( ) method

Saves or sends an XML document as text generated from an X-document object.

Return type: LOGICAL

Applies to: X-document object handle

Syntax

```
SAVE ( mode , { file | stream | stream-handle | memptr | longchar } )
```

mode

A character expression that evaluates to one of the following: "FILE", "STREAM", "STREAM-HANDLE", "MEMPTR", or "LONGCHAR". The *mode* indicates whether the XML target is a file, a stream, a MEMPTR, or a LONGCHAR variable.

file

A character expression that represents the absolute or relative pathname of a new file created to save the XML text. If you specify a relative pathname, it is relative to the current working directory of the underlying operating system's file system. If a file with the specified name already exists, the AVM verifies that the file is writeable and overwrites the file. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

stream

A character expression that represents the name of an ABL stream. If *stream* is "", the AVM saves the document to the ABL session unnamed stream.

stream-handle

A character expression that represents a stream object handle.

memptr

A MEMPTR variable to contain the saved XML text in memory. The SAVE method allocates the required amount of memory and sets the size of the variable. You must release the memory later with a SET-SIZE( ) = 0.

longchar

A LONGCHAR variable to contain the saved XML text in memory.

If the LONGCHAR variable's code page is fixed (that is, set using the FIX-CODEPAGE statement) and the fixed code page is not compatible with the encoding specified in the XML document's ENCODING attribute, the SAVE( ) method returns an error and the XML document is not saved to the LONGCHAR.

If the LONGCHAR variable's code page is not fixed, the AVM converts from the XML document's code page to the -cpinternal code page and saves it to the LONGCHAR variable.

Note: The SAVE( ) method allows some non-printing characters to be written to XML documents. These characters (for example, #x1, #x1F) are not part of the XML Version 1.0 standard, but are allowed in order to support ABL applications.

The following code fragments demonstrate the use of the SAVE( ) method:

```
/* Saves the current tree under hDoc as memo.xml. */
DEFINE STREAM mystream.
OUTPUT STREAM mystream to memo.xml.
hdoc:SAVE("stream","mystream").

OR

/* Saves the current tree under hDoc as memo.xml. */
hdoc:SAVE("file","memo.xml").

OR

/* Saves the current tree under hDoc in memory referred to by mymem. */
DEFINE VARIABLE hDoc  AS HANDLE NO-UNDO.
DEFINE VARIABLE mymem AS MEMPTR NO-UNDO.
. . .
hDoc:SAVE("memptr", mymem).
```

See also

ENCODING attribute

SAVE-FILE( ) method

Saves the current contents of the editor widget to a specified text file and sets the widget's MODIFIED attribute to FALSE.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
SAVE-FILE ( filename )
```

filename

A character-string expression of the absolute or relative pathname of a file. If you specify a relative pathname, it is relative to the current working directory. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

If the save is not successful, it does not change the value of the MODIFIED attribute. If the save is successful, the method returns TRUE.

In Windows, this method writes out text files with a carriage return character and a line feed character terminating each line of text. In all other interfaces, this method writes out text files with a carriage return character terminating each line of text. Also in Windows, **RETURN** key input writes out as `x0d0d0a` and in all other interfaces, writes out as `x0d0a`.

SAVE-ROW-CHANGES( ) method

Saves changes from one row of a ProDataSet temp-table to the associated data source.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
buffer-handle:SAVE-ROW-CHANGES ( [ buffer-index | buffer-name
  [ , skip-list [ , no-lobs ] ] ] )
```

buffer-handle

A handle to a before-image buffer in a ProDataSet object.

buffer-index

An integer expression that specifies the index of the buffer in the data source's buffer list. The default value is 1.

buffer-name

A CHARACTER expression that evaluates to the name of the buffer in the data source.

skip-list

An optional character expression that evaluates to a comma-separated list of field names for fields that should not be assigned after a new row is created (that is, fields to skip). For example, a key field or other fields assigned a value by a CREATE database trigger.

no-lobs

A logical expression indicating whether to ignore BLOB and CLOB fields in the save operation. If TRUE, BLOB and CLOB fields are ignored during the save operation. If FALSE, BLOB and CLOB fields are saved along with the other fields. The default value is FALSE (that is, BLOB and CLOB fields are included in the save operation).

The data source must be attached before calling this method. If there is no data source, or special processing is needed, you must write the code to save the changes instead of using the SAVE-ROW-CHANGES( ) method.

The AVM saves ProDataSet buffer changes to the associated data source based on the current MERGE-BY-FIELD attribute and PREFER-DATASET attribute settings.

When MERGE-BY-FIELD is TRUE, the AVM merges changes from a ProDataSet temp-table buffer to the associated data source on a field-by-field basis. When MERGE-BY-FIELD is FALSE, the AVM does not merge changes on a field-by-field basis.

Note: Merging a large number of changes from a ProDataSet object to the data source on a field-by-field basis is slower than saving changes buffer-by-buffer.

Before saving any changes, the AVM compares the before-image of the ProDataSet temp-table buffer or field, saved while tracking changes for the buffer, to the corresponding buffer or field in the data source to determine whether the data in the data source has changed since being read.

If the data in the data source has not changed, the AVM copies the ProDataSet buffer or field to the data source. If the data in the ProDataSet buffer or field was deleted, the AVM deletes the data from the data source.

If the data in the data source has changed, the AVM saves the ProDataSet buffer changes based on the current PREFER-DATASET attribute setting. When PREFER-DATASET is TRUE, the AVM copies the data from ProDataSet buffer or field to the data source regardless of any changes made to the data in the data source since the data was read. When PREFER-DATASET is FALSE, the AVM does not copy the data from the ProDataSet buffer or field to the data source. The AVM copies the conflicting data from the data source to the ProDataSet buffer or field, instead, and sets the ERROR attribute and DATA-SOURCE-MODIFIED attribute to TRUE.

If the AVM encounters an error, it sets the value of the ERROR attribute to TRUE for the associated ProDataSet object handle, Temp-table object handle, and Buffer object handle.

Note: SAVE-ROW-CHANGES ignores schema delete validation, but does run any DELETE triggers.

SAVE-WHERE-STRING attribute

Returns the WHERE clause used to find the database buffer identified by the buffer-index or buffer-name in a data-source object.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Data-source object handle

Syntax

```
SAVE-WHERE-STRING ( buffer-index | buffer-name )
```

buffer-index

The 1-based index of the buffer in the list of buffers for the data-source object.

buffer-name

The name of a before-image table in the list of buffers for the data-source object. The default is the first (or only) buffer in the data-Source buffer list.

SAX-PARSE( ) method

Performs a single-call parse of an XML document associated with a SAX-reader object.

Return type: None

Applies to: SAX-reader object handle

Syntax

```
SAX-PARSE ( )
```

Returns when one of the following occurs:

- The parser reaches the end of the document.
- A callback invokes RETURN ERROR.
- An error occurred because the parser could not start or could not continue parsing.

While SAX-PARSE( ) is running, it can invoke callbacks, for different elements and events that occur during the execution of the parser. Callbacks are optional, and you do not need to define all the callbacks. The AVM invokes the corresponding callback, if defined, at the appropriate time. Callbacks can reside in one of the following::

- As callback procedures in a procedure file whose handle has been assigned to the HANDLER attribute.
- As callback methods in an object-oriented ABL class whose reference has been assigned to the EVENT-HANDLER-OBJECT attribute.

Note: SAX-PARSE( ) does not have a return value. To detect if an error has occurred, add NO-ERROR to the call and when it returns, check ERROR-STATUS:ERROR.

SAX-PARSE-FIRST( ) method

Initializes and begins a progressive-scan parse of an XML document associated with a SAX-reader object.

Return type: None

Applies to: SAX-reader object handle

Syntax

```
SAX-PARSE-FIRST ( )
```

To continue a progressive-scan parse, call SAX-PARSE-NEXT( ) repeatedly.

When SAX-PARSE-FIRST( ) executes, the AVM invokes the StartDocument callback, if there is one. SAX-PARSE-FIRST( ) raises an ABL error if the parser fails, for any reason, to start parsing the XML source.

If SAX-PARSE-FIRST( ) is called on a source file already being parsed with SAX-PARSE-NEXT( ), the parser reinitializes the parsing process.

SAX-PARSE-FIRST( ) can be called with SAX-reader in any state.

Note: SAX-PARSE-FIRST( ) does not have a return value. To detect if an error has occurred, add NO-ERROR to the call and when it returns, check ERROR-STATUS:ERROR.

SAX-PARSE-NEXT( ) method

Continues a progressive-scan parse of an XML document associated with a SAX-reader object.

Return type: None

Applies to: SAX-reader object handle

Syntax

```
SAX-PARSE-NEXT ( )
```

SAX-PARSE-NEXT( ) returns after the parser finds the next XML token in the XML source and the AVM invokes the corresponding callback, if it exists.

When SAX-PARSE-NEXT( ) starts, an error is raised if the PARSE-STATUS attribute is anything other than SAX-RUNNING. This might occur if:

- SAX-PARSE-NEXT( ) is called before SAX-PARSE-FIRST( ) is called
- STOP-PARSING( ) is called

Eventually, SAX-PARSE-NEXT( ) fails to find another XML token (assuming the parse does not stop early). When this happens, the AVM sets the PARSE-STATUS attribute to SAX-COMPLETE. If SAX-PARSE-NEXT( ) is called at that point, an error is raised.

Note: SAX-PARSE-NEXT( ) does not have a return value. To detect if an error has occurred, add NO-ERROR to the call and when it returns, check `ERROR-STATUS:ERROR`.

To stop a progressive-scan parse, you can refraining from calling SAX-PARSE-NEXT( ). However, when you detect that it is time to stop, you will most likely be in a callback, so you need a way to communicate that it is time to stop to the driver (where SAX-PARSE-NEXT( ) is called). A convenient way to do that is to call `SELF:STOP-PARSING( )`, since that causes the AVM to set the value of PARSE-STATUS to SAX-COMPLETE.

SCHEMA-CHANGE attribute

Relaxes the requirement for exclusive access to a database in order to make the following schema changes online:

- Add new sequences.
- Add new tables, as well as any associated fields, indexes, and database triggers (which must be added within the same transaction).
- Add new fields to an existing table. (You cannot define ASSIGN triggers for new fields while the database is online.)
- Add new inactive indexes to an existing table.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SESSION system handle

Set this attribute to "NEW OBJECTS" to activate this feature; and to the empty string ("") or the Unknown value (?) to deactivate this feature.

SCHEMA-LOCATION attribute

Determines the list of namespace/location pairs of an XML Schema file to validate against.

Data type: CHARACTER

Access: Readable/Writable

Applies to: X-document object handle, SAX-reader object handle

Contains the XML Schema file namespace/location pairs list for the object that the parser is using to validate against. The value pairs are separated by white space. The values can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

The default is an empty string ("").

The syntax for the list is the same as the attribute schemaLocation in an XML document. The first member of each pair is the namespace and the second member is the location where to find to an appropriate XML Schema file. The list is written as pairs of target namespace and locations where each value is separated by white space, using the following format:

Syntax

```
"namespace1 location1 namespace2 location2 namespace3 location3"
```

Here is an example assignment to this attribute:

```
hXdoc:SCHEMA-LOCATION = "http://www.example.com com.xsd
                         http://www.example.org org.xsd"
```

Note that namespace and XML Schema file locations specified programatically with this attribute take precedence over namespaces or schemas declared in XML documents or imported elements.

To clear the schema location list set SCHEMA-LOCATION to an empty string ("").

The previous example could also be achieved invoking the ADD-SCHEMA-LOCATION( ) method twice:

```
hXdoc:ADD-SCHEMA-LOCATION("http://www.example.com", "com.xsd").
hXdoc:ADD-SCHEMA-LOCATION("http://www.example.org", "org.xsd").
```

SCHEMA-MARSHAL attribute

Specifies the amount of schema information to marshal for a temp-table parameter. The temp-table may be an independent temp-table or a member of a ProDataSet object.

If the attribute is not specified, the amount of schema information specified by the value of Temp-table Schema Marshal (-ttmarshal) is used. The default value of `-ttmarshal` is 0, which corresponds to the `FULL` setting for the `SCHEMA-MARSHAL` attribute and includes all schema information for the temp-table parameter.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Temp-table object handle

The following table lists the SCHEMA-MARSHAL attribute values.

Table 104: SCHEMA-MARSHAL attribute values

Attribute value	Description
FULL	Includes all schema information for a temp-table parameter
MIN	Minimizes schema information for a temp-table parameter
NONE	Excludes schema information for a temp-table parameter

If set to the Unknown value (?), the default value is FULL.

Note: If you specify both the SCHEMA-MARSHAL attribute and the MIN-SCHEMA-MARSHAL or NO-SCHEMA-MARSHAL attribute for an individual temp-table, the AVM uses the attribute you most recently specified. The MIN-SCHEMA-MARSHAL and NO-SCHEMA-MARSHAL attributes are supported only for backward compatibility. Use the SCHEMA-MARSHAL attribute instead.

Setting this attribute overrides the setting of the Temp-table Schema Marshal (`-ttmarshal`) startup parameter for an individual temp-table parameter. For more information about this startup parameter, see *Startup Command and Parameter Reference*.

See also

MIN-SCHEMA-MARSHAL attribute, NO-SCHEMA-MARSHAL attribute

SCHEMA-PATH attribute

A delimiter-separated list of directory paths used to locate the XML Document Type Definition (DTD) associated with a particular XML document. It is searched if the XML document contains a relative path to the DTD.

Data type: CHARACTER

Access: Readable/Writable

Applies to: SAX-reader object handle, X-document object handle

The default is the Unknown value (?).

To separate directory paths, use a comma, or a delimiter character that your operating system recognizes.

To separate directory names and filenames within a path, use the UNIX format (forward slashes) or the standard format for your operating system.

SCHEMA-PATH can include HTTP URLs, but they might slow your application since accessing a DTD over the Web might take significantly longer than doing so over a file network.

SCHEMA-PATH cannot contain ABL procedure libraries (`.pl` files).

File pathnames cannot contain embedded commas. The AVM replaces delimiter characters with commas, so the resulting SCHEMA-PATH can be accessed using the ENTRY function.

Pathnames can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

Note: When the AVM searches for the DTD, the AVM concatenates each entry in SCHEMA-PATH with SYSTEM-ID (an attribute in DOM and a parameter of the ResolveEntity callback in SAX), which indicates the system ID of the external DTD. SYSTEM-ID might be as simple as a filename or as complex as a relative path.

For more information on accessing XML documents using the Document Object Model (DOM) or Simple API for XML (SAX) interfaces, see *Use XML with ABL Applications*.

SCREEN-LINES attribute

The number of display lines available in the window, in character units.

Data type: DECIMAL

Access: Read-only

Applies to: WINDOW widget

SCREEN-VALUE attribute

The data value in the screen buffer associated with the widget.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget (cell), COMBO-BOX widget, EDITOR widget, FILL-IN widget, LITERAL widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

Note that setting the SCREEN-VALUE attribute does not affect the record buffer. To apply the updated value to the record buffer you must explicitly assign the field or variable. Likewise, assigning the record buffer does not affect the screen buffer. To display a value in the record buffer, you must explicitly assign it to the SCREEN-VALUE attribute or implicitly move it to the screen buffer using a DISPLAY or UPDATE statement.

For combo boxes, this attribute returns the screen buffer value of the combo box fill-in. If no item in the list is selected or the list is empty, this attribute returns the Unknown value (?). Setting this attribute to an item in the list deselects the previously selected item and assigns the value of the selected item to the fill-in screen buffer. For SIMPLE and DROP-DOWN combo boxes, if the new value in the fill-in is not an item in the list, the fill-in screen buffer is set to the new value and no item in the list is selected. For DROP-DOWN-LIST combo boxes, if the new value in the fill-in is not an item in the drop down list, the AVM ignores the value and displays a warning message.

Note: When a selection-list has the MULTIPLE attribute, setting the SCREEN-VALUE does not clear existing selections. See the MULTIPLE attribute, for more information on how multi-selection operates.

For browse cells, screen values are applied to the buffers automatically when the user leaves the row. If the browse has the NO-ASSIGN option specified in the DEFINE BROWSE statement or it is a dynamic browse, then you must apply the screen values.

Changing the SCREEN-VALUE attribute for a browse column is useful in setting the cells in a non-database browse column.

Note: If you assign a value to the SCREEN-VALUE attribute of a widget and display the widget using a DISPLAY or UPDATE statement, the value you assigned appears as the initial value.

In Windows, SCREEN-VALUE applies to the regular editor and to the large editor.

SCROLL-BARS attribute

Indicates whether scroll bars appear in a window.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WINDOW widget

If the SCROLL-BARS attribute is TRUE, scroll bars appear for the window when a user resizes the window to a size smaller than virtual height or width of the window. The RESIZE attribute controls the ability to resize a window. The VIRTUAL-HEIGHT-CHARS, VIRTUAL-WIDTH-CHARS, VIRTUAL-HEIGHT-PIXELS, and VIRTUAL-HEIGHT-PIXELS attributes control the virtual or maximum size of a window.

SCROLL-TO-CURRENT-ROW( ) method

Scrolls a browse (if necessary) to bring the currently selected row into view. If the browse supports multiple selections, then SCROLL-TO-CURRENT-ROW( ) brings the most recently selected row into view.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SCROLL-TO-CURRENT-ROW ( )
```

The position of the scrolled row is the first row in the browse viewport, unless the current row is already visible. In this case, the current row remains in the original position. If the row is successfully scrolled into view (or if the scroll is unnecessary), the method returns TRUE.

SCROLL-TO-ITEM( ) method

Scrolls a selection list so that the specified item appears at the top of the list.

Return type: LOGICAL

Applies to: SELECTION-LIST widget

Syntax

```
SCROLL-TO-ITEM ( list-item | list-index )
```

list-item

A character-string expression that specifies a single value in the selection list.

list-index

An integer expression that specifies the ordinal position (first, second, third, etc.) of an entry in the selection list.

If the method is successful, it returns TRUE.

SCROLL-TO-SELECTED-ROW( ) method

Scrolls a browse (if necessary) to bring a specified selected row into view.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SCROLL-TO-SELECTED-ROW ( n )
```

n

An integer expression that specifies a selected row within the browse.

The AVM maintains a numbered list of selected rows, starting at 1. When the SCROLL-TO-SELECTED-ROW(*n*) method is encountered, the AVM searches this list to find the *n*th selected row. If the row is successfully scrolled into view (or if the scroll is unnecessary), the method returns TRUE.

SCROLLABLE attribute

The scrolling capabilities of a frame or a dialog box.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget

If the SCROLLABLE attribute is TRUE, the frame or dialog box can be bigger than the display space allotted to it (that is, it is scrollable). If SCROLLABLE is FALSE, the frame or dialog box must fit within the allotted display space; it cannot be made to scroll. The default value is TRUE. The VIRTUAL-HEIGHT-CHARS, VIRTUAL-HEIGHT-CHARS, VIRTUAL-HEIGHT-CHARS, and VIRTUAL-HEIGHT-CHARS attributes control the virtual or maximum size of a frame or dialog box.

SCROLLBAR-HORIZONTAL attribute

Indicates whether a horizontal scroll bar appears in an editor or a selection list.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: EDITOR widget, SELECTION-LIST widget

If the SCROLLBAR-HORIZONTAL attribute is TRUE, a horizontal scroll bar appears on the bottom edge of the widget. For an editor widget, horizontal scrolling is always enabled whether or not a horizontal scroll bar is enabled. For a selection list, the scroll bar must be enabled to scroll the list. The default value is FALSE.

Note: If the SCROLLBAR-HORIZONTAL attribute is set to TRUE, then WORD-WRAP is automatically set to FALSE. Likewise, if you set the WORD-WRAP attribute to TRUE, then SCROLLBAR-HORIZONTAL is automatically set to FALSE.

You can set this attribute only before the widget is realized.

SCROLLBAR-VERTICAL attribute
(Graphical interfaces only)

Indicates whether a vertical scroll bar appears in a browse, editor or a selection list.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget, EDITOR widget, SELECTION-LIST widget

If the SCROLLBAR-VERTICAL attribute is TRUE, a vertical scroll bar appears on the right side of the widget. For an editor widget, vertical scrolling is always enabled whether or not a vertical scroll bar is enabled. For a selection list, the scroll bar must be enabled to scroll the list. The default value is FALSE.

For browses, the SCROLLBAR-VERTICAL attribute defaults to TRUE unless the DEFINE BROWSE statement Browse Options phrase includes the NO-SCROLLBAR-VERTICAL option. When the vertical scroll bar appears, it appears on the right side of the browse.

For editors and selection lists, you can set this attribute only before the widget is realized.

SEAL( ) method

As part of a successful application-performed authentication operation, you can invoke this method on an unsealed client-principal in order to seal the object using the domain access code you pass as a parameter. Sealing the client-principal begins a login session to assert the identity it represents. You can then pass the sealed object as input to the SECURITY-POLICY:SET-CLIENT( ) method method or the SET-DB-CLIENT function to set the identity for the current ABL session and one or more of its database connections using a single sign-on (SSO) operation.

This method returns TRUE if successful and FALSE otherwise.

Return type: LOGICAL

Applies to: Client-principal object handle

Syntax

```
SEAL ( domain-access-code )
```

domain-access-code

A case-sensitive character expression containing the clear text access code that you have defined for the user's domain. The AVM uses this access code to generate the message authentication code (MAC) with which to seal the client-principal object. The AVM converts this access code to UTF-8 before using it, which ensures a consistent value regardless of code page settings. A matching domain with the specified access code must be registered in a trusted domain registry in order to successfully validate the identity that is sealed in this client-principal object.

Caution: The sealed client-principal created by this method is not usable if no trusted domain registry contains the same combination of domain name and access code. Also, note that given any client-principal that is sealed with it, this domain access code provides session and database access when validated using a single sign-on (SSO) operation against a trusted domain registry that contains a matching domain with the same access code. Therefore, Progress Software corporation **strongly** recommends that you take steps to hide or otherwise protect the domain access code in your ABL code from access (hacking) by unauthorized users.

Note: You only need to call the SEAL( ) method if your ABL application successfully performs its own authentication of the user's credentials and therefore needs to initialize and seal a client-principal to represent the authenticated user identity and begin a user login session. The domain that you use to initialize the client-principal is typically configured with either the built-in `_extsso` authentication system or a user-defined authentication system that is enabled for SSO-only operations to validate and set a user identity. For all other supported authentication systems, such as `_oeusertable`, `_oslocal`, or user-defined authentication systems that are enabled for user authentication using an ABL authentication plugin, you can pass an appropriately initialized unsealed client-principal object directly as input to the SET-CLIENT( ) method or SET-DB-CLIENT function to authenticate the identity, seal the client-principal object, create the login session, and set the identity for the ABL session or database connection in a single user authentication operation.

You must validate and set values for the following attributes on the client-principal before you can seal the object with this method:

- USER-ID attribute
- DOMAIN-NAME attribute
- SESSION-ID attribute

The AVM raises a run-time error if:

- You do not appropriately set these attributes.
- The client-principal object is already sealed.
- You invoke SEAL() with an encoded domain access code. (You can only call SEAL() with a clear text domain access code.)

You can seal a client-principal object only once per user login session. You can then use the VALIDATE-SEAL( ) method to validate the seal whenever necessary.

Once sealed, you can no longer set any attributes or user-defined properties for the object.

The method also sets the SEAL-TIMESTAMP attribute with the time stamp for when the client-principal object was sealed, and sets the LOGIN-STATE attribute to `"LOGIN"`.

The method also checks the LOGIN-EXPIRATION-TIMESTAMP attribute. If the client-principal object expires before you can seal it, the AVM sets the LOGIN-STATE attribute to "EXPIRED" and the method returns FALSE.

Calling this method generates a login audit event and creates an audit record for the event in all connected audit-enabled databases according to each database's current audit policy settings.

The following code fragment illustrates how to use the SEAL( ) method:

```
DEFINE VARIABLE hCP     AS HANDLE    NO-UNDO
DEFINE VARIABLE key     AS CHARACTER NO-UNDO.
DEFINE VARIABLE lResult AS LOGICAL   NO-UNDO.
. . .
CREATE CLIENT-PRINCIPAL hCp.
. . .
lResult = hCP:SEAL(key).
```

Note: You typically call this method on an unsealed client-principal object in the INITIAL state only as a result of a **successful** application-performed user authentication in order to seal the object in the LOGIN state. If you want to invalidate an unsealed client-principal object in the INITIAL state because the identity it represents has failed an application-performed user authentication operation, call the AUTHENTICATION-FAILED( ) method, which seals the client-principal in the FAILED state. If you want to invalidate, and terminate the login session for, a **sealed** client-principal object that is in the LOGIN state, call the LOGOUT( ) method, which sets the sealed client-principal to the LOGOUT state.

See also

AUTHENTICATION-FAILED( ) method, LOGIN-EXPIRATION-TIMESTAMP attribute, LOGIN-STATE attribute, LOGOUT( ) method, SEAL-TIMESTAMP attribute, SET-CLIENT( ) method, SET-DB-CLIENT function, VALIDATE-SEAL( ) method

SEAL-TIMESTAMP attribute

Returns the time stamp (as a DATETIME-TZ value) for when the client-principal object was sealed in the LOGIN state. If the client-principal object is not sealed, this attribute returns the Unknown value (?).

Data type: DATETIME-TZ

Access: Read-only

Applies to: Client-principal object handle

See also

LOGIN-STATE attribute, SEAL( ) method, VALIDATE-SEAL( ) method

SEARCH( ) method

Searches for a specified string in an editor widget starting from the current text cursor position. For graphical interfaces only, the string is highlighted when found.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
SEARCH ( string , flag )
```

string

The character-string expression for which to search.

Note: You cannot specify wildcard characters in the search string.

flag

An integer expression that specifies the type of search to be performed.

The *flag* expression determines what type of search to perform. The following table lists the flag values that correspond to each search type.

Table 105: SEARCH flag values

Type of search	Flag value
FIND-NEXT-OCCURRENCE	1
FIND-PREV-OCCURRENCE	2
FIND-CASE-SENSITIVE	4
FIND-WRAP-AROUND	16
FIND-SELECT	32

For a single search operation, you cannot specify both FIND-NEXT-OCCURRENCE and FIND-PREV-OCCURRENCE. Any other combination of these flags is valid. To do multiple searches, you add the flag values. For example, you can specify FIND-PREV-OCCURRENCE and FIND-WRAP-AROUND by adding their flag values, 2 and 16, to invoke SEARCH(*string*, 18).

If the operation is successful, the method returns TRUE.

SELECT-ALL( ) method

Selects all rows, or a range of rows, in a query connected to the browse.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SELECT-ALL ( [
  starting-row-table1 , starting-row-table2, . . .
  ending-row-table1, ending-row-table2 . . .
  ] )
```

starting-row-table1

A variable of type ROWID representing the first row in the first table to select.

starting-row-table2

A variable of type ROWID representing the first row in the second table to select.

ending-row-table1

A variable of type ROWID representing the last row in the first table to select.

ending-row-table2

A variable of type ROWID representing the last row in the second table to select.

If you not specify parameters, the SELECT-ALL method selects all rows.

If you specify the starting row and the ending row, the SELECT-ALL method selects all rows between the starting row and the ending row inclusive.

If the query is a join, a ROWID for each table in the query can be specified for the starting row and the ending row. A maximum of 40 parameters is allowed which allows the user to specify a 20-table join, 20 ROWIDs for the starting row, and 20 ROWIDs for the ending row.

SELECT-FOCUSED-ROW( ) method

Selects the row that currently has focus in a browse widget, even if it is not currently displayed.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SELECT-FOCUSED-ROW ( )
```

This method repositions the query to that row and copies the record into the database buffer. The browse automatically scrolls to the selected row. You can use this method after a REPOSITION statement to position a query to a selected row.

SELECT-NEXT-ROW( ) method

Deselects all currently selected rows in a browse and selects the row after the deselected row.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SELECT-NEXT-ROW ( )
```

This method also repositions the query to the new row and copies the record into the database buffer. The browse automatically scrolls to the selected row if it is out of view.

This method is intended for use with a browse that supports the selection of a single row at a time (MULTIPLE attribute is set to FALSE). If more than one row is selected when you execute this method, all of the selected rows are deselected and the record after the last selected row becomes the selected row.

SELECT-PREV-ROW( ) method

Deselects a currently selected row in a browse and selects the row before the deselected row.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SELECT-PREV-ROW ( )
```

This method also repositions the query to the new row and copies the record into the database buffer. The browse automatically scrolls to the selected row if it is out of view.

This method is intended for use with a browse that supports the selection of a single row at a time (MULTIPLE attribute is set to FALSE). If more than one row is selected when you execute this method, all of the selected rows are deselected and the record before the last selected row becomes the selected row.

SELECT-ROW( ) method

Selects the specified row if it is currently in the browse viewport. In a single-select browse, the previously selected row is deselected. No rows are deselected in a multiple-select browse.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SELECT-ROW ( n )
```

n

An integer expression specifying the ordinal position of a row within the browse.

This method also repositions the query to that row and copies the record into the database buffer.

SELECTABLE attribute
(Graphical interfaces only)

Indicates whether a widget is selectable for direct manipulation at run time.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

If the SELECTABLE attribute is TRUE, users can select and deselect the widget (that is, activate SELECTION and DESELECTION events for the widget). You must also set the SELECTABLE attribute to TRUE for the RESIZABLE attribute to take effect, allowing the user to resize the widget.

Note: Setting the SELECTABLE attribute to TRUE enables direct manipulation events for the widget. These events take precedence over all other events. This effectively prevents data entry using the widget until all direct manipulation events are disabled (that is, MOVABLE, RESIZABLE, and SELECTABLE are all set to FALSE). Also, vertical scrollbars are disabled until no direct manipulation can occur (that is, MOVABLE, RESIZABLE, and SELECTABLE are all set to FALSE).

SELECTED attribute
(Graphical interfaces only)

Indicates whether a widget is selected (highlighted).

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

This attribute can be set in two ways—when the widget's SELECTABLE attribute is TRUE and the user selects the widget, or by setting the SELECTED attribute to TRUE from the AVM whether or not its SELECTABLE attribute is TRUE. Although setting SELECTED to TRUE from the AVM highlights the widget, this does not activate a SELECTION event for the widget.

SELECTION-END attribute

The offset of the first character after the end of the currently selected text in the widget.

Data type: INTEGER

Access: Read-only

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

If no text is currently selected, SELECTION-END has the Unknown value (?).

If the editor is not already realized and you reference its SELECTION-END attribute, the AVM realizes the widget.

In Windows, both the regular editor and the large editor support SELECTION-END.

SELECTION-START attribute

The offset of the first character of the currently selected text in the widget.

Data type: INTEGER

Access: Read-only

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

If the editor is not already realized and you reference its SELECTION-START attribute, the AVM realizes the widget.

In Windows, both the regular editor and the large editor support SELECTION-START.

SELECTION-TEXT attribute

The currently selected text in the widget.

Data type: CHARACTER

Access: Read-only

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

You can read this attribute to access the text the user has selected. To change or remove the currently selected text, use the `REPLACE-SELECTION-TEXT( )` method.

If the editor is not already realized and you reference its `SELECTION-TEXT` attribute, the AVM realizes the widget.

Note: You cannot use this attribute to access the text of a `FILL-IN` widget whose `PASSWORD-FIELD` attribute is set to TRUE.

SENSITIVE attribute

Indicates whether a widget can receive input focus or events. Indicates whether certain ABL objects can receive events.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, Server socket object handle, SLIDER widget, Socket object handle, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

For widgets, if the SENSITIVE attribute is TRUE, the user can give input focus to the widget or can select, move, or resize it (if other attributes are set). A field-level widget must be specified in a frame before you can set the SENSITIVE attribute. The ENABLE statement implicitly sets this attribute to TRUE, and the DISABLE statement sets it to FALSE.

If the READ-ONLY attribute is TRUE for the widget, the SENSITIVE attribute has no effect except to grey out the widget in some environments.

You can use the SENSITIVE attribute to reset the MODIFIED attribute. For example, when the screen-value of a fill-in field is changed, MODIFIED is set to TRUE. If you then set SENSITIVE to FALSE and then to TRUE, you enable the fill-in for further input and the MODIFIED attribute is reset to FALSE.

For the socket and server socket objects, the SENSITIVE attribute indicates whether the object can receive events. The default value of this attribute is TRUE for socket and server socket objects.

If the SENSITIVE attribute is set to FALSE for the socket object, the AVM will not execute the READ-RESPONSE procedure for the socket even if the READ-RESPONSE event occurs.

If the SENSITIVE attribute is set to FALSE for the server socket object, the AVM will stop accepting connections on the port associated with the server socket.

Note: If an application knows it will not receive data on a socket during some period of time, it should set this attribute to FALSE. This allows the application to run more efficiently since the AVM does not monitor the socket if its SENSITIVE attribute is set to FALSE. Data can still be written to an insensitive socket object. When the attribute is set to TRUE, the AVM checks the socket for data.

SEPARATORS attribute
(Graphical interfaces only)

Indicates whether the AVM displays the row and column separators of a browse widget.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget

If SEPARATORS is TRUE, row and column separators appear in the widget. Otherwise, they do not. This attribute can be initialized with the SEPARATORS or NO-SEPARATORS option of the DEFINE BROWSE statement.

SEPARATOR-FGCOLOR attribute
(Graphical interfaces only)

Sets the color of a browse's separators.

Data type: INTEGER

Access: Readable/Writable

Applies to: BROWSE widget

If you set a browse's SEPARATOR-FGCOLOR attribute and the separators appear, they have the color you specified.

SERIALIZE-HIDDEN attribute

Indicates whether a temp-table field or ProDataSet name is written when the temp-table or ProDataSet is serialized, for example into JSON or XML.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Buffer-field object handle, ProDataSet object handle

This attribute's purpose overlaps with the XML-NODE-TYPE attribute. Because of this overlap, the attributes interact as follows:

- The WRITE-XML( ) method always uses the XML-NODE-TYPE attribute value. If you set a value for the XML-NODE-TYPE attribute, it keeps that value regardless of how you set the SERIALIZE-HIDDEN attribute.
- If you do not set the XML-NODE-TYPE attribute and set the SERIALIZE-HIDDEN attribute to TRUE, the AVM sets XML-NODE-TYPE to `"HIDDEN"`.
- If you do not set the XML-NODE-TYPE attribute and set the SERILIAZE-HIDDEN attribute to FALSE, the AVM sets XML-NODE-TYPE to `"ELEMENT"`.
- If you do not set either attribute, the AVM sets XML-NODE-TYPE to `"ELEMENT"` and sets SERIALIZE-HIDDEN to FALSE.

This attribute does not apply to database buffer fields.

For more information about using this attribute, see the documentation on working with JSON.

See also

WRITE-JSON( ) method, WRITE-XML( ) method, XML-NODE-TYPE attribute

SERIALIZE-NAME attribute

Indicates the name of a ProDataSet, a temp-table, a temp-table buffer, or a temp-table buffer-field object as it should appear when serialized, for example into JSON or XML.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer object handle, Buffer-field object handle, ProDataSet object handle, Temp-table object handle

This attribute's purpose overlaps with the XML-NODE-NAME attribute. Because of this overlap, the attributes interact as follows:

- The READ-XML( ) and WRITE-XML( ) methods always use the XML-NODE-NAME attribute value. If you set a value for the XML-NODE-NAME attribute, it keeps that value regardless of how you set the SERIALIZE-NAME attribute.
- If you do not set the XML-NODE-NAME attribute and set the SERIALIZE-NAME attribute, the AVM sets XML-NODE-NAME equal to SERIALIZE-NAME.
- If you do not set either attribute, the AVM sets both to the ABL object name.

Use this attribute when the serialized name either contains invalid characters for an ABL name or the serialized name is an ABL keyword.

This attribute does not apply to database buffers and fields.

For more information about using this attribute, see the documentation on working with JSON.

See also

READ-JSON( ) method, READ-XML( ) method, WRITE-JSON( ) method, WRITE-XML( ) method, XML-NODE-NAME attribute

---

SERIALIZE-ROW( ) method

Serializes a temp-table buffer's current row in either JSON or XML format as a string, or in JSON format as a `Progress.Json.ObjectModel.JsonObject` object.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
SERIALIZE-ROW ( target-format , target-type
  , { file | stream | stream-handle | memptr | longchar | JsonObject }
  [ , formatted [ , encoding [ , omit-initial-values
  [ , omit-outer-object ] ] ] ] )
```

target-format

A CHARACTER expression that specifies the format of the output. Valid values are `"JSON"` and `"XML"`.

target-type

A CHARACTER expression that specifies the target type for the output. Valid values are `"FILE"`, `"STREAM"`, `"STREAM-HANDLE"`, `"MEMPTR"`, `"LONGCHAR"`, and `"JsonObject"`.

All values except `"JsonObject"` specify a target for a JSON or XML string.

When you specify "JsonObject", *target-format* must be "JSON". Otherwise, the SERIALIZE-ROW( ) method generates an error message and returns FALSE.

file

A CHARACTER expression that specifies the name of a file to which the AVM writes the buffer's current row. You can specify an absolute pathname or a pathname relative to the current working directory. If a file with the specified name already exists, the AVM verifies that the file is writeable and overwrites the file.

stream

A CHARACTER expression that specifies the name of a stream. If you specify the empty string (""), the AVM writes the buffer's current row to the default unnamed output stream. For WebSpeed, write the buffer's current row to the WebSpeed-defined output stream (WEBSTREAM).

For more information about using ABL unnamed output streams, see the DEFINE STREAM statement reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces*.

stream-handle

A HANDLE variable that specifies a stream object handle.

memptr

A MEMPTR variable to contain the buffer's current row in memory. If you do not specify the encoding parameter, the AVM encodes the text written to the MEMPTR as UTF-8. This method allocates the required amount of memory for the MEMPTR and sets the size of the variable. When you are finished using the MEMPTR, you must free the associated memory, by executing SET-SIZE(*memptr*) = 0 on the MEMPTR.

longchar

A LONGCHAR variable to contain the buffer's current row in memory.

The AVM saves the buffer's current row to the LONGCHAR variable in the code page that corresponds to the character encoding you specify in the *encoding* option. If you do not specify a character encoding for the LONGCHAR, the AVM saves the LONGCHAR variable in UTF-8.

If the LONGCHAR variable's code page is fixed (that is, set using the FIX-CODEPAGE statement) and the fixed code page is not equivalent to the character encoding you specify in the *encoding* option, the SERIALIZE-ROW( ) method generates an error message and returns FALSE. The buffer's current row is not saved to the LONGCHAR.

JsonObject

A reference to a Progress.Json.ObjectModel.JsonObject object to which the AVM writes the buffer's current row. Any prior contents of the specified *JsonObject* are removed.

The *JsonObject* must be a valid instance of type Progress.Json.ObjectModel.JsonObject. If not, the SERIALIZE-ROW( ) method generates an error message and returns FALSE.

When you specify a *JsonObject* parameter, *target-format* must be "JSON". Otherwise, the SERIALIZE-ROW( ) method generates an error message and returns FALSE.

formatted

An optional LOGICAL expression where TRUE directs the AVM to format the output in a hierarchical manner using extra white space, carriage returns, and line feeds. The default value is FALSE.

If you specify the Unknown value (?), the method uses the default value of FALSE.

The *formatted* option is ignored for *JsonObject* targets.

encoding

An optional CHARACTER expression that specifies the name of the character encoding the AVM uses to write the output. The default encoding is "UTF-8".

When *target-format* is "JSON", the encoding name must specify a Unicode transformation format. Valid values are "UTF-8", "UTF-16", "UTF-16BE", "UTF-16LE", "UTF-32", "UTF-32BE", and "UTF-32LE".

When *target-format* is "XML", the encoding name must be an Internet Assigned Numbers Authority (IANA) name supported by the ABL XML Parser. For a list of supported IANA encodings and their corresponding ABL code pages, see the following table in the ENCODING attribute reference entry.

Note: If you specify the empty string ("") or the Unknown value (?), the AVM uses the default encoding of UTF-8.

---

omit-initial-values

An optional LOGICAL expression where TRUE directs the AVM to exclude temp-table fields containing their initial values from the output, and FALSE directs the AVM to include all temp-table field data in the output. The default value is FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

When working with large ProDataSets, omitting fields containing their initial values can yield smaller output size, more efficient network transfers, and performance gains with the READ-JSON( ) and SERIALIZE-ROW( ) methods.

This behavior applies both to temp-table fields that have the default initial value for its data type, and for fields that have an initial value set with the ABL INITIAL option.

Although using the *omit-initial-values* option can give your application performance and resource use improvements, you must be sure that the consumers of the generated JSON output can correctly handle it. The ABL READ-JSON( ) and READ-XML( ) methods always populate created records with initial values from the temp-table or ProDataSet definition. Other applications might not do this.

omit-outer-object

An optional LOGICAL expression that indicates whether the outer-most object in JSON output is included. TRUE directs the AVM to remove the outer-most object on output. FALSE directs the AVM to include the outer-most object in the output. The default is FALSE.

If you specify the Unknown value (?), the method uses the default value of FALSE.

The *omit-outer-object* option is only valid when *target-format* is "JSON". If *omit-outer-object* is TRUE when *target-type* is "XML", the SERIALIZE-JSON( ) method generates an error message and returns FALSE.

Notes

- You cannot call SERIALIZE-ROW( ) on a database buffer handle.
- SERIALIZE-ROW( ) can serialize fields with data types based on class-based objects to JSON format. See WRITE-JSON( ) method for a detailed list of restrictions that apply to any class-based objects that are to be written to JSON.

- When executing the method, there is no guarantee as to the contents of the buffer involved after the method finishes executing.

See also

Progress.Json.ObjectModel.JsonObject class, READ-JSON( ) method, READ-XML( ) method, WRITE-JSON( ) method, WRITE-XML( ) method

SERVER attribute

The server handle to one of the following:

- An application server on which:
  - An asynchronous request is running
  - The specified remote persistent, single-run, or singleton procedure is running
  - You are running a remote dynamic procedure or user-defined function
  - WebClient application files are accessed
- A server that hosts a Web service:
  - On which an asynchronous Web service operation is running
  - To which a Web service procedure object is bound

Data type: HANDLE

Access: Readable/Writable

Applies to: Asynchronous request object handle, Call object handle, CODEBASE-LOCATOR system handle, Procedure object handle

For an asynchronous request object handle, the SERVER attribute refers to the application server where an asynchronous procedure call is active, or it refers to the server that hosts a Web service where an asynchronous operation is executing.

For a procedure handle, the SERVER attribute is read-only and valid only on a proxy persistent, single-run, or singleton procedure handle that references an active remote procedure running in the context of an application server or that references a port type procedure object in the context of a Web service (that is, where the handle PERSISTENT and PROXY attributes are both set to TRUE). Thus, for a persistent, single-run, or singleton procedure, the attribute refers to the application server where it runs, and for a port type procedure object, the attribute refers to the server that hosts the Web service. Otherwise, the SERVER attribute is set to the Unknown value (?).

For the call object handle, the SERVER attribute is both readable or writable and specifies the handle to an application server containing logic you want to invoke dynamically.

Note: Do not confuse SERVER and IN-HANDLE. SERVER indicates a handle to an application server containing logic you want to invoke dynamically. IN-HANDLE indicates a handle to a running persistent procedure containing logic (internal procedures and user-defined functions) you want to invoke dynamically.

For the CODEBASE-LOCATOR system handle, the SERVER attribute is read-only and returns the server handle to a connected application server for accessing WebClient application files. Valid only if LOCATOR-TYPE is `"AppServer"`.

To check the validity of a handle, use the VALID-HANDLE function.

SERVER-CONNECTION-BOUND attribute
(PAS for OE only)

Indicates if the current application server session is bound to a particular client application.

Data type: LOGICAL

Access: Read-only

Applies to: SESSION system handle

This attribute is valid only if the REMOTE attribute is TRUE.

If the SERVER-CONNECTION-BOUND attribute is TRUE, the current session is bound to a client application. Otherwise, it is FALSE.

On a session-free application server, if the SERVER-CONNECTION-BOUND-REQUEST attribute is set to FALSE to unbind the connection, the connection remains bound and the SERVER-CONNECTION-BOUND attribute remains TRUE as long as remote persistent procedures remain active for the connection.

On a session-managed application server, the SERVER-CONNECTION-BOUND attribute is always set to TRUE.

SERVER-CONNECTION-BOUND-REQUEST attribute
(PAS for OE only)

Tells an application server session running on a session-free application server to bind or unbind its current client connection.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

This attribute is valid only if the REMOTE attribute is TRUE and the operating mode is stateless.

When set to TRUE, requests that the application server session makes its connection bound to the current client connection identified by the SERVER-CONNECTION-ID attribute.

When set to FALSE, requests that the application server session makes its connection unbound from the currently bound client connection pending the deletion of all remote persistent procedures running in the session. When all remote persistent procedure for the connection have been deleted, the application server session becomes available to service a different client connection. The SERVER-CONNECTION-BOUND attribute for the session is also set to FALSE when the application server session becomes available.

In state-free operating mode, any attempt to set this attribute raises a WARNING condition in the application server session, which writes a message to the application server log file, and the value remains unchanged. You can handle the WARNING condition by including the NO-ERROR option in the statement that attempts to set the value, and checking ERROR-STATUS:NUM-MESSAGES for a value greater than zero. In state-free operating mode, this attribute always has the Unknown value (?).

Note: This attribute has no effect on application server sessions running in a session-managed mode.

SERVER-CONNECTION-CONTEXT attribute
(PAS for OE only)

An application-determined value that you set within an application server session. The AVM passes this value to each application server session that executes a request on behalf of the client connection identified by the SERVER-CONNECTION-ID attribute.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SESSION system handle

This attribute is valid only if the REMOTE attribute is TRUE.

When a client application requests a connection with an application server, the application server creates an area to store this value for the connection. The initial value is the Unknown value (?).

This attribute, while available in session-managed and session-free operating modes, has practical application only on a session-managed application server, where more than one application server session can service the same client connection. This value is available to any Connect procedure, Activate procedure, Deactivate procedure, or Disconnect procedure that you have configured for the application server, as well as any application procedure. Thus, each application server session that services a client connection can pass context information to the next.

For an application server session, the AVM sets the SERVER-CONNECTION-CONTEXT attribute to the Unknown value (?) each time a new connection is assigned to the process. If the application server operating mode is session-managed, the application server session procedure can also reset this attribute to an application-specific value. However, any such value does not last beyond the current client connection within the current application server session. Thus, application server sessions running on a session-managed application server cannot pass information among themselves using this attribute.

In session-free operating mode, any attempt to set this attribute raises a WARNING condition in the application server session, which writes a message to the application server log file, and the value remains unchanged. You can handle the WARNING condition by including the NO-ERROR option in the statement that attempts to set the value, and checking `ERROR-STATUS:NUM-MESSAGES` for a value greater than zero. In session-free operating mode, this attribute always has the Unknown value (?).

SERVER-CONNECTION-ID attribute
(PAS for OE only)

Returns the run-time connection ID of the current client connection assigned to this application server session.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

This attribute is valid only if the REMOTE attribute is TRUE.

This value is assigned by the application server when an application server accepts a connection request from a client application. The application server sessions use the connection ID as an identifier when they log any information associated with the connection. This same connection ID is available to the application server session using the SERVER-CONNECTION-ID attribute and to the connected ABL client using the CLIENT-CONNECTION-ID attribute on the server handle connected to this application server.

The value of the connection ID is guaranteed to be globally unique for all time within a single computer network. Connection IDs can be compared to each other strictly for equality, but other types of comparisons are irrelevant.

The AVM ensures that the SERVER-CONNECTION-ID attribute for each application server agent is set to the connection ID for the connection that is assigned to the application server session. Each time a new connection is assigned to an application server session, a new value is assigned to the SERVER-CONNECTION-ID attribute.

This attribute is available to any Connect procedure or Disconnect procedure that you have configured for the application server. It maintains the same value for these and all other application server procedures executed on behalf of the same connection.

If the application server operating mode is session-managed, the application server resets the SERVER-CONNECTION-ID attribute for each application server session to the ID of the connection each time it executes a request. The connection ID for a bound ABL session remains the same until the server process becomes unbound and receives a request from a new unbound connection.

If the application server operating mode is session-free, this value has no meaning.

Note: You can identify client context for a user login session on an application server running in any operating mode (including session-free) using the ClientContextId property of the Progress.Lang.OERequestInfo class.

SERVER-OPERATING-MODE attribute
(AppServer only; PAS for OE only)

Returns the operating mode of the current Progress Application Server (PAS) for OpenEdge or AppServer session.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

This attribute is valid only if the REMOTE attribute is TRUE.

The possible values for this attribute when used with the AppServer include:

- `"State-reset"`
- `"State-aware"`
- `"Stateless"`
- `"State-free"`

This is the value of the operatingMode property set for this AppServer in the `ubroker.properties` file.

The possible values for this attribute when used with PAS for OpenEdge include:

- `"Session-managed"`
- `"Session-free"`

Because an instance of the Progress Application Server (PAS) for OpenEdge can run sessions in both modes simultaneously, the operating mode is not set in a properties file. The operating mode is specified through a parameter on the CONNECT( ) method on the client.

See also

CONNECT( ) method (AppServer), CONNECT( ) method (Progress Application Server for OpenEdge)

SESSION-END attribute

Sets the end of a logical WebSpeed session. Intended for internal use only.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

SESSION-ID attribute

Specifies the user login session ID for the user represented by the client-principal object. This attribute must be set before the associated unsealed client-principal object can be sealed in the LOGIN state using the SEAL( ) method, or using a SET-DB-CLIENT function or SECURITY-POLICY:SET-CLIENT( ) method call that performs a user authentication operation.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

If you specify the Unknown value (?) or the empty string (""), the AVM generates a run-time error.

For login session auditing, set this attribute to a unique value. You can have OpenEdge set a 22-character, Base64-encoded UUID value for this attribute by initializing the client-principal object using the INITIALIZE( ) method.

Note: OpenEdge provides no inherent test of uniqueness for this value.

You can also set this attribute to one of the following values:

- The ClientContextId property on the Progress.Lang.OERequestInfo class, which you can use to uniquely identify both identity and application context for a user login session in either a session-free or session-managed application server application
- The SERVER-CONNECTION-ID attribute on the SESSION system handle, to map the user ID of the user login session to the connection ID of an application server session (supported for backward compatibility in session-managed applications)
- The universally unique identifier (UUID) generated by the GENERATE-UUID function
- Any other globally unique value provided by your application.

Once the client-principal object is sealed, this attribute is read-only.

See also

INITIALIZE( ) method, SEAL( ) method, SET-CLIENT( ) method, SET-DB-CLIENT function

SET-ACTOR( ) method

Sets the ACTOR attribute for this SOAP header entry.

Return type: LOGICAL

Applies to: SOAP-header-entryref object handle

Syntax

```
SET-ACTOR ( character )
```

character

A character variable containing the URI of the SOAP actor. The actor can be used to indicate the recipient of a SOAP header element.

Call this method once you have associated the XML with a SOAP header entry using the SET-NODE( ) method.

Note: In SOAP Version 1.2, the ROLE attribute replaces the SOAP Version 1.1 ACTOR attribute. If you use SET-ACTOR( ) to set the ACTOR attribute, the AVM automatically assigns the value to the ROLE attribute when accessing a SOAP Version 1.2 Web service.

SET-APPL-CONTEXT( ) method

Sets (and records) the application context for the current session in each connected audit-enabled database whose current audit policy has this audit event enabled. Application context provides meaningful information about the conditions under which an application audit event occurred.

This method returns a Base64 character string that specifies the universally unique identifier (UUID) of the primary index for all audit event records generated by this method for this application context. This UUID is recorded in all subsequent audit event records until you either clear this application context or set a different application context. The UUID is 22 characters in length (the two trailing Base64 pad characters are removed).

Return type: CHARACTER

Applies to: AUDIT-CONTROL system handle

Syntax

```
SET-APPL-CONTEXT ( event-context
  [ , event-detail [ , audit-custom-detail ] ] )
```

event-context

A character expression that specifies the context for the audit event. The value of this expression cannot exceed 200 characters. You can also use this value as an alternate index for querying the audit event record.

If you specify the Unknown value (?), the AVM generates a run-time error.

event-detail

An optional character expression that specifies additional audit detail. The value of this expression cannot exceed 10,000 characters.

audit-custom-detail

An optional character expression that specifies additional user detail. The value of this expression cannot exceed 10,000 characters.

The UUID is saved as the APPL-CONTEXT-ID attribute value for each connected audit-enabled database.

There can be only one active application context per session at any one point in time. To set a different application context for the session, you can:

- Call the CLEAR-APPL-CONTEXT( ) method, to clear the current application context, and then call the SET-APPL-CONTEXT( ) method with the new application context.
- Call the SET-APPL-CONTEXT( ) method with the new application context. If there is an existing application context in effect, the AVM clears the existing application context before setting the new application context.

Calling this method generates an audit event, and creates an audit record for the event in all connected audit-enabled databases according to each database's current audit policy settings.

The following code fragment illustrates how to use the SET-APPL-CONTEXT( ) method:

```
DEFINE VARIABLE name AS CHARACTER NO-UNDO.
DEFINE VARIABLE id   AS CHARACTER NO-UNDO.
. . .
id = AUDIT-CONTROL:SET-APPL-CONTEXT("Payroll app", "fica calculation", name).
. . .
AUDIT-CONTROL:LOG-AUDIT-EVENT(34123, "payroll.fica.calc").
. . .
AUDIT-CONTROL:CLEAR-APPL-CONTEXT.
```

See also

APPL-CONTEXT-ID attribute, CLEAR-APPL-CONTEXT( ) method

SET-ATTRIBUTE( ) method

Adds a new attribute to an element. If an attribute with the same name is already present, its value is replaced with the specified value.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
SET-ATTRIBUTE ( name , value )
```

name

A character expression that represents the attribute name.

value

A character expression that represents the attribute value.

The following example creates the following line in the hDoc output .xml file:

```
<Customer Id="54" Name="Second Skin Scuba"/>
```

```
CREATE X-DOCUMENT hDoc.
CREATE X-NODEREF hRoot.
CREATE X-NODEREF hNoderef.

hDoc:CREATE-NODE(hRoot,"root","ELEMENT").
hDoc:APPEND-CHILD(hRoot).
...
hDoc:CREATE-NODE(hNoderef,"Customer","ELEMENT").
hNoderef:SET-ATTRIBUTE("Id","54").
hNoderef:SET-ATTRIBUTE("Name","Second Skin Scuba").
hRoot:APPEND-CHILD(hNoderef).
```

SET-ATTRIBUTE-NODE( ) method

Associates an XML ATTRIBUTE node with the referenced X-noderef object handle.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
SET-ATTRIBUTE-NODE ( attr-node-handle )
```

attr-node-handle

A valid X-noderef handle that represents an XML ATTRIBUTE node created with the CREATE-NODE-NAMESPACE( ) or CREATE-NODE( ) method.

SET-BLUE-VALUE( ) method
(Graphical interfaces only)

Specifies the blue component of an entry in the color table. You cannot change a static color with this method. Before changing the color, you must make the color dynamic using the SET-DYNAMIC( ) method. For an example, see r-colhan.p in the COLOR-TABLE system handle entry.

Return type: LOGICAL

Applies to: COLOR-TABLE system handle

Syntax

```
SET-BLUE-VALUE ( index , blue-value )
```

index

An integer expression that specifies an entry in the color table.

blue-value

An integer expression that specifies the blue RGB component of an entry in the color table. The value must be in the range 0 to 255.

If the operation is successful, the method returns TRUE.

SET-BREAK( ) method

Sets a breakpoint for a debugging session.

Return type: LOGICAL

Applies to: DEBUGGER system handle

Syntax

```
SET-BREAK ( [ procedure [ , line-number ] ] )
```

procedure

A character expression that specifies the name of the procedure in which you want to set the breakpoint. The specified procedure does not have to exist at the time the breakpoint is set. If you do not specify *procedure*, the method sets the breakpoint at the next executable line of the current procedure.

line-number

An integer expression that specifies the line number in *procedure* (based at line 1 of the debug listing) where you want to set the breakpoint. A positive integer greater than or equal to 1 represents a line number in the specified *procedure* file. Zero (0) or a negative integer value represents the first executable line of the main procedure block in the specified *procedure* file. If you do not specify *line-number*, the method sets the breakpoint at the first executable line of *procedure* file. If *line-number* is greater than the last executable line number, the method sets the breakpoint at the last executable line of *procedure*. If *line-number* does not specify an executable line, the method sets the breakpoint at the next executable line after the line specified by *line-number*.

If the Debugger is initialized, this method returns TRUE. Otherwise, it returns FALSE with no effect. For more information, see the reference entry for the DEBUGGER system handle.

Note: To use this method, you must have the Application Debugger installed in your OpenEdge environment.

Note that the Debugger sets breakpoints on **physical lines**—not statements. If you invoke DEBUGGER:SET-BREAK( ) on a line that contains other executable statements, all the other statements on that line execute before the breakpoint occurs on the next executable line. This is true whether the statements appear on the same line before or after the invocation of the SET-BREAK( ) method.

If you invoke DEBUGGER:SET-BREAK (*procedure* , *line-number*) on the same line that is specified by *procedure* and *line-number*, the specified line executes the first time without breaking. The breakpoint occurs only on the second and succeeding executions of the line.

Note: You cannot set a watchpoint programmatically using the DEBUGGER system handle. A watchpoint is a form of breakpoint which tells the Debugger to interrupt program execution when the value of a variable, buffer field, or attribute reference changes.

SET-BUFFERS( ) method

Binds all buffers for a query or dynamic ProDataSet object at the same time. Any buffers previously added to the ProDataSet object are removed.

Use the ADD-BUFFER( ) method to add one buffer to the object, without affecting the other buffers, if any.

Return type: LOGICAL

Applies to: ProDataSet object handle, Query object handle

Syntax

```
SET-BUFFERS ( buffer [ , buffer ] ... )
```

buffer

A handle to a buffer, or a CHARACTER expression that evaluates to the name of a **DEFINED** buffer (you cannot specify the default buffer), that the AVM searches for at run time.

Note: The maximum number of buffers for a query or ProDataSet is 200.

Note: Searching for a buffer using a handle is more efficient than a character expression. The AVM resolves a character expression at run time by searching in the current routine for a static buffer with that name. If not found, the AVM searches the enclosing main procedure. If still not found, the AVM searches up through the calling programs of the current routine, and their main procedures. Since a handle uniquely identifies the buffer, no such search is required.

The following is an example when the parameter is a handle to a buffer:

```
my-query-handle:SET-BUFFERS(BUFFER customer:handle).
```

The following is an example when the parameter is a CHARACTER expression:

```
DEFINE BUFFER bCustomer FOR Customer.
my-query-handle:SET-BUFFERS("bCustomer").
```

SET-CALLBACK( ) method

Associates a method within a class instance, or an internal procedure within a persistent procedure, with an ABL callback event.

Return type: LOGICAL

Applies to: Buffer object handle, Procedure object handle, ProDataSet object handle, Query object handle, Socket object handle, Server socket object handle, RUN statement

Syntax

```
SET-CALLBACK ( callback-name , routine-name [ , routine-context ] )
```

callback-name

A quoted string or character expression representing the name of a callback. The *callback-name* is not case-sensitive.

- For a ProDataSet buffer object handle, this can be "BEFORE-FILL", "AFTER-FILL", "BEFORE-ROW-FILL", "AFTER-ROW-FILL", "ROW-CREATE", "ROW-DELETE", "ROW-UPDATE", "FIND-FAILED", or "SYNCHRONIZE".
- For a Procedure object handle, when calling a Web service, this can be either "REQUEST-HEADER" or "RESPONSE-HEADER".
- For a ProDataSet object handle, this can be "BEFORE-FILL" or "AFTER-FILL".
- For a Query object handle, this can be "OFF-END".
- For a Socket object handle, this can be "READ-RESPONSE".
- For a Server Socket object handle, this can only be "CONNECT".
- For SOAP header callback events, this can only be "REQUEST-HEADER" for a SOAP request event or "RESPONSE-HEADER" for a SOAP response event. The corresponding procedure or method associated with the request or response event is invoked. For a SOAP request or response message, the associated procedure or method handler provides access to the SOAP request or response header during the execution of the Web service operation.

You can also use the APPLY-CALLBACK( ) method to apply a callback procedure for a Buffer, ProDataset, or Query object. You cannot use the APPLY-CALLBACK( ) method for a Procedure, Socket or Server-socket object.

routine-name

A quoted string or character expression representing the name of a method or an internal procedure that resides within *routine-context*. If calling this on a procedure handle, routine-name can only be an internal procedure.

routine-context

An object reference for a class instance or a handle to a persistent procedure that contains the method or internal procedure specified by *routine-name*. If not specified, and the routine is executed within a procedure, THIS-PROCEDURE is used as the routine context. If not specified, and the routine is executed within a class instance, THIS-OBJECT is used as the routine context.

If *callback-name* is not a valid callback, or *routine-context* is not a valid object reference or handle, this method returns FALSE; otherwise, it returns TRUE. If the SET-CALLBACK( ) method is specified but cannot be invoked, or it fails, no method or procedure is executed as part of the callback event.

Notes

- The callback method or procedure must be public.
- If *routine-name* represents a class method that is associated with an ABL event as a callback, and the method is overridden in a derived class, the overridden method is run when the callback is invoked.
- To clear a callback, pass the Unknown (?) value or an empty string as the *routine-name* parameter. The following code fragment shows clearing a callback by passing the empty string:

```
BUFFER <buffer-name>:HANDLE:SET-CALLBACK ("After-Row-Fill", "").
```

- When used with a class instance, the SET-CALLBACK( ) method creates an internal reference to the instance. As a result, the instance cannot be garbage collected while the callback is in effect. If you clear it, the object may be garbage collected if there are no more references to it.
- For a Socket object:
  - The callback method or procedure signature must not have any parameters. Overloaded methods are allowed, but only the method with no parameters is invoked on the READ-RESPONSE event.
  - The method return type should be VOID. Any value returned from the method is ignored.
- For a server socket object:
  - The callback handler signature takes a single input parameter of type HANDLE. The handle value is the socket handle for the client connection.
  - The callback method return type should be VOID. Any value returned from the method is ignored.

See also

APPLY-CALLBACK( ) method, GET-CALLBACK-PROC-CONTEXT( ) method, GET-CALLBACK-PROC-NAME( ) method, OFF-END event, ProDataSet events, READ-RESPONSE event, CONNECT event

SET-CALLBACK-PROCEDURE( ) method

Associates an internal procedure with an ABL callback event.

Return type: LOGICAL

Applies to: Buffer object handle, Procedure object handle, ProDataSet object handle, Query object handle

Syntax

```
SET-CALLBACK-PROCEDURE ( callback-name , internal-procedure
                       [ , procedure-context ] )
```

callback-name

A quoted string or character expression representing the name of a callback. The *callback-name* is not case-sensitive.

- For a ProDataSet buffer object handle, this can be "BEFORE-FILL", "AFTER-FILL", "BEFORE-ROW-FILL", "AFTER-ROW-FILL", "ROW-CREATE", "ROW-DELETE", "ROW-UPDATE", "FIND-FAILED", or "SYNCHRONIZE".
- For a Procedure object handle, when calling a Web service, this can be either "REQUEST-HEADER" or "RESPONSE-HEADER".
- For a ProDataSet object handle, this can be "BEFORE-FILL" or "AFTER-FILL".
- For a Query object handle, this can be "OFF-END".

You can also use the APPLY-CALLBACK( ) method to apply a callback procedure for an object.

internal-procedure

A quoted string or character expression representing the name of an internal procedure that resides within *procedure-context*.

procedure-context

A handle to a procedure that contains the internal procedure specified by *internal-procedure*. If not specified, THIS-PROCEDURE is used as the procedure context.

This method returns FALSE if the *callback-name* is not valid or if *procedure-context* is not a valid handle; returns TRUE otherwise. If this method is specified but cannot be invoked, or it fails, no procedure will be executed as part of the callback event.

Notes

- If this method is called multiple times, the new values passed replace the previously set values.
- To remove the callback procedure associated with a Web service procedure, invoke this method with a valid *callback-name* and an empty string ("") for the *internal-procedure*. The AVM will not generate an error if an attempt is made to remove a callback procedure and a callback is not associated with the callback.
- For more information on events, see the Handle-based Object Events Reference.

See also

APPLY-CALLBACK( ) method, GET-CALLBACK-PROC-CONTEXT( ) method, GET-CALLBACK-PROC-NAME( ) method, OFF-END event, ProDataSet events

SET-CLIENT( ) method

Sets the identity, using an unsealed or a sealed client-principal object, for the current ABL session, and for all connected and unlocked OpenEdge database connections.

For an unsealed client-principal object (in the INITIAL state)—this method performs a user authentication operation on the user identity asserted by the object. If successful, the method seals the client-principal and assigns the user identity to the ABL session.

For a sealed client-principal object (in the LOGIN state)—this method performs a single sign-on (SSO) operation to validate the user identity represented by the object. If successful, the method assigns the user identity to the ABL session.

If the ABL session identity is set successfully, the method performs an SSO operation to validate and (if successful) set the user identity for each OpenEdge database connection in the ABL session that has not been previously set using the SET-DB-CLIENT function or the SETUSERID function.

Note: Any subsequent calls to either the SETUSERID or SET-DB-CLIENT functions override the user identity for any database connections set by SET-CLIENT( ) and locks these connection identities from any further change by this method unless you unlock each connection. For more information on unlocking a database connection identity, see the SET-DB-CLIENT function reference entry.

Return type: LOGICAL

Applies to: SECURITY-POLICY system handle

Syntax

```
SET-CLIENT ( client-principal-handle )
```

client-principal-handle

A handle to a client-principal object. If the client-principal object is unsealed, it must be initialized with the attribute values required by the SEAL( ) method in addition to any PRIMARY-PASSPHRASE attribute value required to authenticate the asserted user identity. If the object is sealed, it must be sealed with a domain access code that is the same as the access code configured for the user's domain stored in the session domain registry and in the domain registry trusted by each database connection to validate the user's identity. If the handle value is set to the Unknown value (?), the method raises a run-time error and the current session and database connection identities remain unchanged.

If the method returns TRUE, user authentication or validation is successful, the specified identity is set for the current ABL session, and OpenEdge attempts to set the identity using SSO for each unlocked OpenEdge database connection. If validation of the identity fails for any unlocked database connection, the method also returns TRUE, but the ERROR-STATUS system handle returns a message for each database connection that failed SSO validation.

If the method returns FALSE, the identity of the session remains unchanged, and OpenEdge does not set the identity for any existing database connections.

Within a transaction on a:

- **Multi-tenant database** — Any attempt to set an identity for the connection that changes the current database tenancy raises a run-time error.
- **Non-multi-tenant database** — As a best practice, Progress Software recommends that you **not** set a new identity for the connection.

To set a session identity through a user authentication operation:

- The client-principal object must be **unsealed** (in the INITIAL state).
- The required attributes must be properly set (see the SEAL( ) method entry).
- The DOMAIN-NAME attribute must be set to the name of an authentication-enabled domain that is registered in the session domain registry. This can include a domain configured with a user-defined authentication

system that has an ABL authentication plugin enabled. For information on OpenEdge support for domains and domain configuration, see *Learn about Identity Management*.

Note: To set a session identity when the authentication system is your ABL application, you must manually authenticate the user credentials for the client-principal object, explicitly call the SEAL( ) method to seal the object, and perform an SSO operation using this method to set the session identity.

If the user authentication operation fails, for auditing purposes, this function implicitly executes the AUTHENTICATION-FAILED( ) method on the client-principal and leaves the previous session and any database connection identities unchanged.

To set a session identity through an SSO operation:

- The client-principal object must be **sealed** and in the LOGIN state.
- The object must be valid according to the session domain registry: the object must be sealed using the access code defined for a registry domain whose name matches the domain name stored in the object.
- The domain in the registry must be configured with an authentication system that supports SSO.

If the LOGIN-STATE attribute for a sealed client-principal object is not set to `"LOGIN"`, the AVM raises a run-time error and the current user identity remains unchanged.

This method also checks the value of the LOGIN-EXPIRATION-TIMESTAMP attribute on the client-principal object. If the timestamp has expired before the method can seal (during user authentication) or validate (during SSO) the object, the method sets the LOGIN-STATE attribute to `"EXPIRED"` and returns FALSE.

This method returns FALSE with messages returned in the ERROR-STATUS system handle when:

- The client-principal object is sealed and the domain configuration is restricted to user authentication operations.
- The client-principal object is sealed and SSO validation of its identity for the current ABL session fails.
- The client-principal object is unsealed and the domain configuration is restricted to SSO operations.
- The client-principal object is unsealed and user authentication of its asserted identity for the current ABL session fails.

Notes

- To rely on the SET-CLIENT( ) method for all identity settings, do not use the SETUSERID function, SET-DB-CLIENT function, or the CONNECT statement to set the connection identity or you must unlock every database connection whose identity has been set with these mechanisms.
- After a user identity is set for a database connection, the AVM uses that identity to determine if the user has permission to access tables and fields in that particular database.
- When a user identity is set for an application with at least one connected audit-enabled database, this method generates an audit event and creates an audit record for the event in every connected audit-enabled database on which the identity was set according to each database's current audit policy settings.
- You can use this method, instead of the SETUSERID function, to set a database connection identity whether or not the user account is in the `_User` table.
- This method does not attempt set the connection identity for the foreign data source of a DataServer connection. However, it does attempt to set the connection identity for the OpenEdge schema holder database.

See also

Client-principal object handle, GET-CLIENT( ) method, SEAL( ) method, SET-DB-CLIENT function, SETUSERID function

SET-COMMIT( ) method
(PAS for OE only)

Directs the transaction object to commit the transaction when the application server session completes the current request and returns execution to the client.

Return type: LOGICAL

Applies to: Transaction object handle

Syntax

```
SET-COMMIT ( )
```

If the operation is successful, the method returns TRUE. If a transaction initiating procedure is not active in the current application server session, this method returns FALSE. You also **cannot** invoke this method after prior invocation of a SET-ROLLBACK( ) method during service of the same client request.

SET-CONNECT-PROCEDURE( ) method

Identifies the name of the procedure that is invoked when a CONNECT event occurs.

Return type: LOGICAL

Applies to: Server socket object handle

Syntax

```
SET-CONNECT-PROCEDURE ( event-internal-procedure [ , procedure-context ] )
```

event-internal-procedure

A quoted string or character expression representing the name of an internal procedure that resides within *procedure-context*. When a client has requested a connection to this port, the specified internal procedure is called.

procedure-context

A handle to a procedure that contains the internal procedure specified by *event-internal-procedure*. If not specified, THIS-PROCEDURE is used as the *procedure-context*.

Returns FALSE if *procedure-context* is not a valid handle, returns TRUE otherwise. If this method is not invoked, or if it fails, no connection procedure will be executed when the CONNECT event occurs.

For more information on connecting sockets, see *OpenEdge Programming Interfaces*.

SET-DYNAMIC( ) method
(Graphical interfaces only)

Sets a color entry to a dynamic or static color.

Return type: LOGICAL

Applies to: COLOR-TABLE system handle

Syntax

```
SET-DYNAMIC ( index , logical-expr )
```

index

An integer expression that specifies an entry in the color table.

logical-expr

A logical expression that specifies the dynamic status of an entry in the color table.

If *logical-expr* is TRUE and sets the entry to a static color if *logical-exp* is FALSE. If the operation is successful, the method returns TRUE.

SET-GREEN-VALUE( ) method
(Graphical interfaces only)

Specifies the green component of an entry in the color table. If the operation is successful, the method returns TRUE.

You cannot change a static color with this method. Before changing the color, you must make the color dynamic using the SET-DYNAMIC( ) method. For an example, see r-colhan.p in the COLOR-TABLE system handle entry.

Return type: LOGICAL

Applies to: COLOR-TABLE system handle

Syntax

```
SET-GREEN-VALUE ( index , green-value )
```

index

An integer expression that specifies an entry in the color table.

green-value

An integer expression that specifies the green RGB component of an entry in the color table. The value must be in the range 0 to 255.

SET-INPUT-SOURCE( ) method

Specifies the source of XML text to be parsed by a SAX-reader object.

Return type: LOGICAL

Applies to: SAX-reader object handle

Syntax

```
SET-INPUT-SOURCE ( mode , { file | memptr | handle | longchar } )
```

mode

A CHARACTER expression evaluating to "FILE", "MEMPTR", "HANDLE", or "LONGCHAR" indicating whether the XML source is a file, a MEMPTR, a WEB-CONTEXT system handle, or a LONGCHAR variable.

file

A CHARACTER expression that indicates the name of a file that contains the source XML text. This can be a relative pathname, an absolute pathname, or an HTTP URL. Any relative pathname is relative to the current working directory. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

memptr

A MEMPTR variable that contains the source XML text. The size of the MEMPTR variable must match the size of the XML text.

handle

A WEB-CONTEXT system handle. In a WebSpeed application, this directs SAX-reader to get the XML source from WEB-CONTEXT.

longchar

A LONGCHAR variable that contains the source XML text. The size of the LONGCHAR variable must match the size of the XML text.

The following checks are not performed by SET-INPUT-SOURCE( ) but are performed by SAX-PARSE( ) at run time:

- Whether or not a file exists and is accessible
- Whether or not a MEMPTR is usable
- Whether or not WEB-CONTEXT's XML attribute is YES

Note: If WEB-CONTEXT is used outside of a WebSpeed environment, the AVM raises a run-time error.

SET-MUST-UNDERSTAND( ) method

Sets the MUST-UNDERSTAND attribute for this SOAP header entry.

Return type: LOGICAL

Applies to: SOAP-header-entryref object handle

Syntax

```
SET-MUST-UNDERSTAND ( logical )
```

logical

A logical variable containing the value of the MUST-UNDERSTAND attribute.

Call this method once you have associated the XML with a SOAP header entry using the SET-NODE( ) method.

SET-NODE( ) method

Replaces the underlying XML of a SOAP-header-entryref object with the DOM XML sub-tree rooted by the X-noderef object.

Return type: LOGICAL

Applies to: SOAP-header-entryref object handle

Syntax

```
SET-NODE ( x-noderef )
```

x-noderef

A variable of type X-noderef that refers to the XML that will become the SOAP-header-entryref object.

The supplied X-noderef object corresponds to the SOAP header entry element. The method performs a deep copy of the X-noderef object and its sub-tree and adds namespace declarations to the SOAP-header-entryref as necessary. For example, if the X-noderef object uses a namespace that is declared in its parent tree by an ancestor of the X-noderef object, that namespace declaration is carried over to the SOAP-header-entryref object.

Care must be taken to distinguish between a SOAP header entry and how it is represented in ABL. A SOAP header is represented by the SOAP-header object. A SOAP header entry is represented by a SOAP-header-entryref object. The SOAP-header object maintains a list of pointers to existing SOAP-header-entryref objects that have been explicitly associated with the SOAP-header object by calling its ADD-HEADER-ENTRY( ) method.

Before you can use the SET-NODE( ) method on the SOAP-header entryref object, it must be associated with its SOAP-header object. Call the SOAP-header object's ADD-HEADER-ENTRY( ) method prior to calling the SET-NODE method( ) or the SET-SERIALIZED method( ) for the first time. Otherwise, you will get a run-time error.

If you call either the SET-NODE( ) or SET-SERIALIZED( ) method twice in a row with the same SOAP-header-entryref, the second call will overwrite the underlying SOAP header entry created by the first call. Although you can use the same SOAP-header-entryref object to create more than one SOAP header entry, you need to call the ADD-HEADER-ENTRY( ) method between calls to these methods to create a new entry on the SOAP-header object's list and associate the new entry with the current content of the SOAP-header-entryref object.

SET-NUMERIC-FORMAT( ) method

Sets the NUMERIC-SEPARATOR and NUMERIC-DECIMAL-POINT attributes simultaneously.

Return type: LOGICAL

Applies to: SESSION system handle

Syntax

```
SET-NUMERIC-FORMAT ( separator , decimal-point )
```

separator

A CHARACTER expression that represents, in formatted text, a number's thousands separator.

The thousands separator cannot be represented by any of the following:

- The characters: `B C D R Z z 0 1 2 3 4 5 6 7 8 9 + - < > ( ) * ?`

  Note: The space character is allowed.

- Any multi-byte character

decimal-point

A CHARACTER expression that represents, in formatted text, a number's decimal point.

The decimal point cannot be represented by any of the following:

- The characters: `B C D R Z z 0 1 2 3 4 5 6 7 8 9 + - < > ( ) * ?`
- The space character
- Any multi-byte character

SET-NUMERIC-FORMAT( ) returns TRUE if the operation is successful.

Note: The values set by the SET-NUMERIC-FORMAT( ) method override the values set by the Thousands Separator (`-numsep`) and Fractional Separator (`-numdec`) startup parameters.

SET-OUTPUT-DESTINATION( ) method

Specifies the target for an XML document that a SAX-writer object creates.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
SET-OUTPUT-DESTINATION ( mode
  , { file | stream | stream-handle | memptr | longchar } )
```

mode

A character expression that evaluates to one of the following: "FILE", "STREAM", "STREAM-HANDLE", "MEMPTR", or "LONGCHAR". The *mode* indicates whether the XML target is a file, a stream, a MEMPTR, or a LONGCHAR variable.

file

A CHARACTER expression that indicates the name of a file to which the XML document is to be written. This can be a relative pathname, an absolute pathname, or an HTTP URL. Any relative pathname is relative to the current working directory. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

stream

A character expression that represents the name of an ABL stream. If stream is "", the AVM saves the document to the unnamed stream of the ABL session.

stream-handle

A character expression that represents a stream object handle.

memptr

A MEMPTR variable to which the XML document is to be written. The MEMPTR will be overwritten and the new size of the MEMPTR variable will match the size of the XML text.

longchar

A LONGCHAR variable to which the XML document is to be written. The LONGCHAR will be overwritten and the new size of the LONGCHAR variable will match the size of the XML text.

When writing an XML document to a LONGCHAR variable, the AVM writes the LONGCHAR variable in the code page of the XML document as determined by the XML document's ENCODING attribute. If the ENCODING attribute is not set, the LONGCHAR variable is saved in UTF-8.

If the LONGCHAR variable's code page is fixed (that is, set using the FIX-CODEPAGE statement), the code page must be equivalent to the encoding specified in the XML document's ENCODING attribute. If not, the START-DOCUMENT( ) method returns an error and the XML document is not saved to the LONGCHAR variable.

Use this method to set the output destination, which is where the object will write the XML document. This method must be called before you call any of the writing methods or they will raise errors. You can only call this method when the object's WRITE-STATUS property is SAX-WRITE-IDLE or SAX-WRITE-COMPLETE. In other words, you cannot set a new output destination while the SAX-writer object is currently writing XML. This method fails and generates an error message if it is called while writing.

The SET-OUTPUT-DESTINATION method does not check if the specified destination is valid. If does not check whether a MEMPTR or LONGCHAR variable is usable, and it does not check whether a file location or stream is accessible. This destination is checked at run time by the START-DOCUMENT method.

When writing to a MEMPTR or LONGCHAR, the method deletes the previous contents and allocates new memory. For example, writing to a MEMPTR is the logical equivalent of using SET-SIZE based upon the document size after calling END-DOCUMENT. This is the same way that the X-document object handles memory.

SET-PARAMETER( ) method (Handle)

Lets you set parameters for one of the following:

- A procedure or user-defined function you want to invoke dynamically
- A Windows DLL routine or UNIX shared library routine you want to invoke dynamically
- An attribute you want to get or set dynamically
- A method you want to invoke dynamically

Return type: LOGICAL

Applies to: Call object handle

Syntax

```
SET-PARAMETER ( parameter-number , data-type , iomode , parameter-value )
```

parameter-number

An integer expression indicating the order of the parameter. Use 1 for the first parameter, 2 for the second parameter, and so on.

data-type

A character expression indicating the data type of the parameter and evaluating to one of the following:

- "CHARACTER"
- "DATASET-HANDLE"
- "DATE"
- "DATETIME"
- "DATETIME-TZ"
- "DECIMAL"
- "HANDLE"
- "INT64"

- "INTEGER"
- "LOGICAL"
- "LONGCHAR"
- "MEMPTR"
- "RAW"
- "ROWID"
- "TABLE-HANDLE"

TABLE-HANDLE can be a handle to a static or dynamic temp-table. Likewise, DATASET-HANDLE can be a handle to a static or dynamic ProDataSet object.

As with non-dynamic programming, you cannot pass a BLOB or CLOB as a parameter to a method.

To manage BUFFER parameters, use HANDLE parameters.

For each parameter, the data type specified by the caller and the callee must be compatible. For example, an ABL application must specify a 64-bit ABL data type (INT64) when calling a DLL routine that expects a 64-bit signed integer. The ABL program cannot declare the ABL variable as LONG or UNSIGNED-LONG and expect the correct behavior. Similarly, if a DLL routine expects a 32-bit value, the ABL program must specify a 32-bit ABL data type (LONG or UNSIGNED-LONG). The ABL program cannot declare the ABL variable as a SHORT or UNSIGNED-SHORT and expect the correct behavior.

When invoking a Windows DLL or UNIX shared library function, ABL DLL data types are valid as *data-type* values. *data-type* specifies the type expected by the DLL or shared library routine parameter. For example, the parameter *data-type* is set to "LONG" if the DLL routine parameter expects "LONG". For a list of valid ABL DLL data types, see the DEFINE PARAMETER statement.

If a DLL data type is specified for a non-DLL invocation, the AVM raises a run-time error.

Note: If you pass an ABL character string to a UNIX shared library routine, you can pass it as a CHARACTER variable or expression. However, if you expect the shared library routine to modify the value, Progress Software Corporation recommends that you pass a MEMPTR containing a null-terminated character string.

---

iomode

A character expression indicating the mode of the parameter and evaluating to one of the following:

- "INPUT"
- "OUTPUT"
- "INPUT-OUTPUT"
- "OUTPUT-APPEND"

For each parameter, the mode specified by the caller and the callee must match.

For parameters of attributes, specify "INPUT".

For TABLE-HANDLE output parameters whose output is to be appended to the table, and in no other case, specify "OUTPUT-APPEND".

If *data-type* is "DATASET-HANDLE" or "TABLE-HANDLE", you can append "-by-reference" to any previously listed *iomode*.

parameter-value

An expression whose type is compatible with *data-type*.

The *parameter-value* argument may represent a determinate or indeterminate array. However, you cannot pass an array by value. You can pass an array only as a variable defined using the DEFINE VARIABLE statement with the EXTENT option. If you include the EXTENT keyword in the *data-type* argument, the AVM ignores it. If you include the array element values in the *parameter-value* argument, ABL generates a compile error.

If *iomode* is "INPUT" or "INPUT-OUTPUT", if the data type of *parameter-value* does not agree with the data type passed, *parameter-value* will automatically be converted to the data type passed during SET-PARAMETER( ) processing.

If *iomode* is "OUTPUT" or "INPUT-OUTPUT", each of the following must be true:

- *parameter-value* must represent a program variable or a NO-UNDO TEMP-TABLE field (perhaps with an array reference).
- *parameter-value* must still be in scope and must still be valid when the dynamic invoke is executed.
- If the output value from the called procedure does not have a data type that matches the data type passed, the output value will be converted to the data type passed.

If *iomode* is "OUTPUT", *parameter-value* is ignored if the ASYNCHRONOUS attribute is TRUE.

SET-PROPERTY( ) method

Creates and sets the value of the specified application-defined property stored in the unsealed client-principal object.

Return type: LOGICAL

Applies to: Client-principal object handle

Syntax

```
SET-PROPERTY ( property-name , property-value )
```

property-name

A character string that specifies the case-sensitive name of an application-defined property stored in the client-principal object. You must enclose this character string in quotes. If you specify the Unknown value (?) or the empty string (""), the AVM raises a run-time error.

property-value

A character expression that contains the value for the specified property. You must enclose this character expression in quotes. If you specify the Unknown value (?) or the empty string (""), the AVM sets the property to that value (and you cannot change it).

If successful, this method returns TRUE. Otherwise, it returns FALSE.

If you call this method more than once per property, the AVM raises a run-time error. This prevents the definition of multiple properties with the same name.

After you seal the client-principal with a user authentication operation (SEAL( ) method, SECURITY-POLICY:SET-CLIENT( ) method, or SET-DB-CLIENT function), you cannot set any new or existing properties for the object. If you call this method for a sealed client-principal object, the AVM raises a run-time error.

Calling this method does not generate an audit event or an audit record.

You can use the GET-PROPERTY( ) method to get the value of a single property stored in the client-principal, or use the LIST-PROPERTY-NAMES( ) method to retrieve a list of the names of all properties stored in the object.

The following code fragment illustrates how to use the SET-PROPERTY( ) method:

```
DEFINE VARIABLE hCP    AS HANDLE  NO-UNDO.
DEFINE VARIABLE val-ok AS LOGICAL NO-UNDO.
. . .
CREATE CLIENT-PRINCIPAL hCP.
hCP:SET-PROPERTY("eye-color", "Blue").
. . .
val-ok = hCP:VALIDATE-SEAL(key).
```

See also

GET-PROPERTY( ) method, LIST-PROPERTY-NAMES( ) method

SET-READ-RESPONSE-PROCEDURE( ) method

Specifies the name of the procedure to invoke when a READ-RESPONSE event occurs.

Return type: LOGICAL

Applies to: Socket object handle

Syntax

```
SET-READ-RESPONSE-PROCEDURE ( event-internal-procedure
                            [ , procedure-context ] )
```

event-internal-procedure

A quoted string or character expression representing the name of an internal procedure that resides within *procedure-context*. When data is available on the socket, the specified internal procedure is called. If not specified, then no read procedure will be executed when the READ-RESPONSE event occurs.

procedure-context

A handle to a procedure that contains the internal procedure specified by *event-internal-procedure*. If not specified, THIS-PROCEDURE is used as the *procedure-context*.

Notes

- Returns FALSE if the *procedure-context* is not a valid handle; returns TRUE otherwise. If this method is not invoked, or it fails, no read procedure will be executed when the READ event occurs.
- In this procedure, the SELF handle identifies the affected socket object. Once you've ultimately deleted the socket object, you should also delete the SELF handle to avoid a memory leak.

SET-RED-VALUE( ) method
(Graphical interfaces only)

Specifies the red component of an entry in the color table. You cannot change a static color with this method. Before changing the color, you must make the color dynamic using the SET-DYNAMIC( ) method. For an example, see r-colhan.p in the COLOR-TABLE system handle entry.

Return type: LOGICAL

Applies to: COLOR-TABLE system handle

Syntax

```
SET-RED-VALUE ( index , red-value )
```

index

An integer expression that specifies an entry in the color table.

red-value

An integer expression that specifies the red RGB component of an entry in the color table. The value must be in the range 0 to 255.

If the operation is successful, the method returns TRUE.

SET-REPOSITIONED-ROW( ) method

Sets the row index where records positioned with the REPOSITION TO ROWID (or RECID) statement are displayed.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SET-REPOSITIONED-ROW ( n , "ALWAYS" | "CONDITIONAL" )
```

n

Indicates the row number where the new record is displayed, 1 being the first row.

"ALWAYS"

Specifies that the REPOSITION TO ROWID statement always uses the indicated row number. If all of the records in a result set fit in the browse viewport, the REPOSITION statement only moves focus to the existing row.

"CONDITIONAL"

Specifies that the REPOSITION TO ROWID statement uses the indicated row number unless the new row is already in the browse viewport. In this case, the REPOSITION statement moves focus to the existing row.

By default, this is the top row in the browse viewport (index 1). If the associated query is defined with the INDEXED-REPOSITION option, the CONDITIONAL option is ignored.

SET-RGB-VALUE( ) method
(Graphical interfaces only)

Specifies a combination of the red, green, and blue values of an entry in the color table. You cannot change a static color with this method. Before changing the color, you must make the color dynamic using the SET-DYNAMIC( ) method. For an example, see r-colhan.p in the COLOR-TABLE system handle entry.

Return type: LOGICAL

Applies to: COLOR-TABLE system handle

Syntax

```
SET-RGB-VALUE ( index , integer-expression )
```

index

An integer expression that specifies an entry in the color table.

integer-expression

An integer expression that specifies the RGB component of an entry in the color table. You can obtain this value from the color property of an ActiveX control, by using the RGB-VALUE function, or by using the GET-RGB-VALUE( ) method.

If the operation is successful, the method returns TRUE.

SET-ROLE( ) method
(SOAP 1.2 only)

Sets the ROLES attribute for a SOAP header entry.

Return type: LOGICAL

Applies to: SOAP-header-entryref object handle

Syntax

```
SET-ROLE ( character )
```

character

A character variable that specifies the URI of a SOAP role. The role can be used to indicate the recipient of a SOAP header element.

Call this method once you have associated the XML with a SOAP header entry using the SET-NODE( ) method.

Note: In SOAP Version 1.2, the SET-ROLE method replaces the SOAP Version 1.1 SET-ACTOR method. A runtime error occurs if the AVM sends a ROLE attribute in a SOAP header to a SOAP Version 1.1 Web service.

SET-ROLLBACK( ) method
(PAS for OE only)

Directs the transaction object to rollback the transaction when the application server session completes the current request and returns execution to the client.

Return type: LOGICAL

Applies to: Transaction object handle

Syntax

```
SET-ROLLBACK ( )
```

If the operation is successful, the method returns TRUE. If a transaction initiating procedure is not active in the current application server session, this method returns FALSE. You also **can** invoke this method after prior invocation of a SET-COMMIT( ) method during service of the same client request.

SET-SAFE-USER( ) method

This method only applies to a Progress Application Server (PAS) for OpenEdge application server.

Sets a multi-tenant database connection's current user identity to the stored safe user identity for the database. If the Safe User ID feature is not enabled, the method returns the Unknown (?) value and the identity of the connected session remains unchanged. If the Safe User ID feature is enabled and the method succeeds in setting the database connection's current user identity to the safe user identity, the method returns TRUE; otherwise it returns FALSE.

Return-type: LOGICAL

Applies to: SECURITY-POLICY system handle

Syntax

```
SET-SAFE-USER ( logical-db-name )
```

logical-db-name

The logical name of the database for which you want to set the safe user identity. The logical database name must be a character string enclosed in quotes, or a character expression.

The following example code shows how to call the SET-SAFE-USER( ) method.

```
DEFINE VARIABLE safeUserSet as LOGICAL NO-UNDO.

safeUserSet = SECURITY-POLICY:SET-SAFE-USER(LDBNAME(1)).
IF safeUserSet
THEN
  MESSAGE "Safe user has been set".
ELSE
  MESSAGE "Safe user was not set".
```

See also

GET-SAFE-USER( ) method

Overview of Safe User ID in *Learn about Identity Management*

SET-SELECTION( ) method

Selects (and highlights) the text in a widget between two specified character offsets.

Return type: LOGICAL

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

Syntax

```
SET-SELECTION ( start-pos , end-pos )
```

start-pos

An integer expression that specifies the offset of the first character to be selected.

end-pos

An integer expression that specifies the offset of the first character after the selection.

This method selects the text that begins at the offset *start-pos* and ends at the offset *end-pos*. If the operation is successful, the method returns TRUE. Otherwise, it returns FALSE.

Note: This operation produces a different result depending on the platform. In Windows GUI platforms, ABL measures character offset positions **between** characters. On non-Windows GUI or character platforms, ABL measures character offset positions **on** characters.

In Windows, both the regular editor and the large editor support SET-SELECTION.

For browses, this method applies only to fill-in and combo-box browse columns.

SET-SERIALIZED( ) method

Sets the SOAP-header-entryref object's underlying XML from serialized XML.

Return type: LOGICAL

Applies to: SOAP-header-entryref object handle

Syntax

```
SET-SERIALIZED ( longchar )
```

longchar

A variable of type LONGCHAR that contains the serialized XML that will become the SOAP-header-entryref object.

Functions the same as SET-NODE( ), except it expects a LONGCHAR whose contents is equivalent to the serialized form of the X-noderef that would be passed to the SET-NODE( ) method. No validation is done by this method. If the contents of the LONGCHAR is not valid for the SOAP message, an error might be raised when the SOAP message is processed. If the LONGCHAR is empty, or the caller passes the Unknown value (?), the header entry will be removed from the SOAP message.

Care must be taken to distinguish between a SOAP header entry and how it is represented in ABL. A SOAP header is represented by the SOAP-header object. A SOAP header entry is represented by a SOAP-header-entryref object. The SOAP-header object maintains a list of pointers to existing SOAP-header-entryref objects that have been explicitly associated with the SOAP-header object by calling its ADD-HEADER-ENTRY( ) method.

Before you can use the SET-SERIALIZED( ) method on the SOAP-header entryref object, it must be associated with its SOAP-header object. Call the SOAP-header object's ADD-HEADER-ENTRY( ) method prior to calling the SET-SERIALIZED method( ) or the SET-NODE method( ) for the first time. Otherwise, you will get a run-time error.

If you call either the SET-SERIALIZED( ) or SET-NODE( ) method twice in a row with the same SOAP-header-entryref, the second call will overwrite the underlying SOAP header entry created by the first call. Although you can use the same SOAP-header-entryref object to create more than one SOAP header entry, you need to call the ADD-HEADER-ENTRY() method between calls to these methods to create a new entry on the SOAP-header object's list and associate the new entry with the current content of the SOAP-header-entryref object.

SET-SOCKET-OPTION( ) method

Sets the specified socket option. TCP supports a number of socket options. Please refer to TCP documentation for a description of these options.

Return type: LOGICAL

Applies to: Socket object handle

Syntax

```
SET-SOCKET-OPTION ( name , arguments )
```

name

A character expression which indicates the name of the socket option to be set.

arguments

A character expression that contains a comma separated list of arguments specific for the option.

The following table describes the options ABL supports.

Table 106: Options for the SET-SOCKET-OPTION( ) method

Option	Description
TCP-NODELAY	An *enable indicator*, which is either TRUE or FALSE.
SO-LINGER	Two comma separated values: • The *onoff indicator*, which is either TRUE or FALSE. • The *linger time*. If the *onoff indicator* is FALSE, the linger time does not need to be provided.
SO-KEEPALIVE	Sets the TCP socket option SO_KEEPALIVE. Set *arguments* to TRUE to turn this option on or to FALSE to turn it off.
SO-REUSEADDR	Sets the TCP socket option SO_REUSEADDR. Set *arguments* to TRUE to turn this option on or to FALSE to turn it off.

Option	Description
SO-RCVBUF SO-SNDBUF	Sets the TCP socket option SO_RCVBUF or SO_SNDBUF. Set *arguments* to the desired size of the buffer. **Note:** Depending on your platform, the value you supply might be increased to the platform's minimum buffer size, decreased to the platform's maximum buffer size, or rounded up to the next multiple of the platform's segment size. For more information, see your platform's documentation.
SO-RCVTIMEO	Sets the timeout length—that is, the number of seconds the socket waits to receive data before timing out. Set *arguments* to the desired timeout value in seconds. If a timeout occurs, READ( ) returns TRUE and the value of BYTES-READ is zero. This is true whether the READ( ) mode is READ-AVAILABLE or READ-EXACT-NUM. For more information on the interaction of READ( ), the READ( ) mode, and SO_RCVTIMEO, see *OpenEdge Programming Interfaces*.

The SET-SOCKET-OPTION( ) method returns TRUE if setting the option succeeded and returns FALSE otherwise. An error can occur if:

- *name* is not an ABL supported socket option
- The arguments supplied for the option are not valid
- The SET-SOCKET-OPTION( ) operation fails

SET-SORT-ARROW( ) method

Sets the sort-arrow indicator and, optionally, the sort-level number for a browse column.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SET-SORT-ARROW ( column-index , ascending [ , n ] )
```

column-index

An INTEGER expression that specifies the column for which the sort-arrow indicator is set. If the column does not exist, the AVM generates a run-time error.

ascending

A LOGICAL expression where TRUE indicates ascending sort order, and FALSE indicates descending sort order.

Set to the Unknown value (?) to remove a sort-arrow indicator that was previously set for this column. Use the CLEAR-SORT-ARROWS( ) method to clear sort-arrow indicators for all columns in the browse.

n

An optional INTEGER expression that specifies the sort level to display at the right of the sort arrow in the column label. Valid values are 1 through 9, inclusive, and the Unknown value (?). If you do not specify a value, or you specify the Unknown value (?), a sort-level number does not appear in the column label. If you specify an invalid value, the AVM generates a run-time error.

You can set two or more columns in a browse to the same sort-level number, and you can leave gaps in the sequence. This allows for rearranging values, as necessary.

The sort-arrow indicator, which consists of an arrow bitmap pointing upward to indicate ascending sort order and pointing downward to indicate descending sort order, is displayed with the column label. It appears at the immediate right of the column label, whether the label is right-justified or left-justified.

The optional sort-level number, if specified, indicates multi-level sorting and appears at the right of the sort arrow in the column label. For example, you can set the sort-level number to show that data in one column of a browse has first been sorted in ascending order and data in another column has then been sorted in descending order.

Note: Setting a sort-arrow indicator and sort-level number on a column does not change the column's sort order or the browse's query. The browse's query is responsible for actually sorting the data.

You can set a sort-arrow indicator and a sort-level number on a browse column before or after the browse widget is realized. Calling this method does not realize the widget.

If the column label text is too wide for the column, the AVM displays as much of the label as possible and clips the rest of the text. If there is no space in the column label to display the sort-arrow indicator, the AVM will not make the column wider at run time; consider increasing the width of the column.

If the browse has one or more stacked column labels, the sort-arrow indicators appear centered vertically within the label header. Note that the sort-arrow indicators are positioned based on the overall height of the label header, not based on the height of any individual column label.

In character interfaces, the ascending sort-arrow indicator is represented by the caret (0x5E) and the descending indicator is represented by the lower case "v" (0x76). These characters are not configurable. The sort-arrow indicator appears at the far right of the column label. This minimizes the chance of the user misreading the arrow as part of the column label. If the column label text is too wide for the column, the AVM displays as much of the label as possible and clips the rest of the text. If the browse has one or more stacked column labels, the sort-arrow indicators appear on the top row of the label header.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

See also

CLEAR-SORT-ARROWS( ) method

SET-WAIT-STATE( ) method

Sets or cancels an ABL wait state that blocks user and system input.

Return type: LOGICAL

Applies to: SESSION system handle

Syntax

```
SET-WAIT-STATE ( state-string )
```

state-string

A character-string expression that sets the wait state.

The value of *state-string* determines the wait state and the type of wait cursor displayed. The valid values are:

- **"GENERAL"** — Displays the system-busy cursor
- **"COMPILER"** — Displays the ABL compiler cursor

Input is blocked and the wait cursor is displayed until the wait state is cancelled. The null string ("") cancels the wait state.

The SET-WAIT-STATE( ) method accepts an arbitrary mouse pointer name (any string which is a valid argument to the LOAD-MOUSE-POINTER( ) method) as an argument, in addition to the "GENERAL", "COMPILER", and "" states. The return value is TRUE if the wait-state is set successfully; otherwise the return value is FALSE.

Note that this method is intended to provide user feedback for lengthy processing that involves no user input, such as compiling procedures, doing a time consuming database lookup, or some long CPU and memory operation like computing the value of p.

This method is not supported in character mode.

If an error occurs from this method, the AVM displays the error and terminates the wait state.

Caution: Be sure that the processing you invoke after setting the wait state is guaranteed to cancel the wait state. Otherwise, the AVM remains in the wait state indefinitely. For example, do not place user input statements, such as SET or UPDATE, between the setting and cancelling of the wait state. Because the user cannot respond to these statements during the wait state, ABL I/O blocks indefinitely, preventing the wait state from being cancelled.

SHOW-IN-TASKBAR attribute
(Windows only)

Determines whether an icon for the window appears in the taskbar and in the task-switching window displayed when **ALT+TAB** is pressed.

Data type: LOGICAL

Access: Readable/Writable

Applies to: WINDOW widget

Applications that display several windows might want only the main window to have an icon on the taskbar.

This attribute defaults to TRUE. If SMALL-TITLE is set to TRUE, the SHOW-IN-TASKBAR attribute will be set to FALSE because, in general, a tool palette should not have an icon in the taskbar. You can override this behavior by setting the SHOW-IN-TASKBAR attribute to TRUE after setting the SMALL-TITLE attribute to TRUE.

If a window that does not appear in the taskbar is minimized, Windows shrinks the window so only the title bar is visible. Windows displays the window at the bottom of the screen. This is standard behavior, but might be unexpected to people who are used to finding minimized windows in the taskbar.

The SHOW-IN-TASKBAR attribute must be set before the window is realized.

The Windows taskbar (through a user property setting) can direct that all minimized windows from the same application be grouped under one taskbar icon. However, the icons for ABL windows whose SHOW-IN-TASKBAR attribute you set to TRUE always appear individually on the taskbar, regardless of taskbar property settings, because ABL does not support this window grouping feature of the taskbar.

.NET forms also have a `ShowInTaskbar` property (provided by the Progress.Windows.Form class in OpenEdge) that corresponds to the SHOW-IN-TASKBAR window attribute. However, .NET forms typically do conform to the taskbar window grouping setting and might well do so when combined with ABL windows in an ABL session, subject to control by the operating system.

SIDE-LABEL-HANDLE attribute

A handle to the side label of a widget.

Data type: HANDLE

Access: Readable/Writeable

Applies to: COMBO-BOX widget, EDITOR widget, FILL-IN widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TEXT widget

For static widgets, this attribute is read-only and the handle accesses a literal widget containing the side label specified when the widget was defined. For dynamic widgets, you can set this handle to a text widget that you create as a side label. You first must create a dynamic text widget to use as a label (assign it a value, row, and column); then assign the handle of the text widget to the SIDE-LABEL-HANDLE attribute of the widget whose label you want to specify or change.

SIDE-LABELS attribute

Indicates whether a frame displays labels to the left of each field.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget

The SIDE-LABELS attribute returns TRUE if the frame displays labels to the left of each field rather than above each field.

Note: For static widgets, colons are appended to the end of side labels by default. To create side labels without colons automatically added, the r-code must be compiled with the `-nocolon` startup parameter in effect. See the *Startup Command and Parameter Reference* for more information.

SIGNATURE-VALUE attribute

Returns the signature value stored in an r-code file.

Data type: CHARACTER

Access: Read-only

Applies to: RCODE-INFO handle

The return value is a Base64-encoded value.

Use this attribute to determine if a procedure changed between different versions of your application.

SINGLE-RUN attribute

The SINGLE-RUN attribute is TRUE when the RUN statement that invokes a procedure is executed with the SINGLE-RUN option. Otherwise, it is FALSE.

Data type: LOGICAL

Access: Read-only

Applies to: Procedure object handle

For more information about the SINGLE-RUN option, see the RUN statement.

SINGLETON attribute

The SINGLETON attribute is TRUE when the RUN statement that invokes a procedure is executed with the SINGLETON option. Otherwise, it is FALSE.

Data type: LOGICAL

Access: Read-only

Applies to: Procedure object handle

For more information about the SINGLETON option, see the RUN statement.

SKIP-DELETED-RECORD attribute

Indicates whether the AVM should skip deleted records when accessing a dynamic query's result list.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Query object handle

SMALL-ICON attribute

Returns the name of the icon loaded by LOAD-SMALL-ICON( ).

Data type: CHARACTER

Access: Read-only

Applies to: WINDOW widget

SMALL-TITLE attribute

Indicates whether the window has a palette-style title bar.

Data type: LOGICAL

Access: Readable/Writable

Applies to: WINDOW widget

This title bar is shorter than a normal Windows title bar, and is commonly used for tool palettes (such as in the AppBuilder) and other auxiliary windows. Windows with small title bars do not have maximize or minimize buttons; they only have close buttons.

The MIN-BUTTON and MAX-BUTTON attributes have no effect on a window with a small title bar and are ignored. The CONTROL-BOX attribute specifies whether the window has a close button and system menu (available by right-clicking on the title bar or by pressing **ALT+SPACE**).

The SMALL-TITLE attribute must be set before the window is realized. The default value of SMALL-TITLE is FALSE.

SOAP-FAULT-ACTOR attribute

The URI of the Web service actor that caused this SOAP fault. If the actor information is not provided, the attribute returns an empty string (" ").

Data type: CHARACTER

Access: Read-only

Applies to: SOAP-fault object handle

Note: In SOAP Version 1.2, the combination of the SOAP-FAULT-NODE and SOAP-FAULT-ROLE attributes replaces the SOAP-FAULT-ACTOR attribute in SOAP Version 1.1.

SOAP-FAULT-CODE attribute

Identifies the SOAP fault code for this SOAP-fault object.

Data type: CHARACTER

Access: Read-only

Applies to: SOAP-fault object handle

SOAP-FAULT-DETAIL attribute

Returns the handle of the SOAP fault detail information associated with this SOAP-fault object.

Data type: HANDLE

Access: Read-only

Applies to: SOAP-fault object handle

SOAP-FAULT-MISUNDERSTOOD-HEADER attribute
(SOAP 1.2 only)

Returns the list of fully qualified names of SOAP headers, if any, for which mandatory processing (as designated by the SOAP-ENV:mustUnderstand attribute) failed. If the SOAP-fault object includes no such failure information, the attribute returns an empty string ("").

Data type: CHARACTER

Access: Read-only

Applies to: SOAP-fault object handle

Note: If the ABL client is connected via SOAP Version 1.1, the SOAP-FAULT-MISUNDERSTOOD-HEADER attribute returns an empty string ("").

SOAP-FAULT-NODE attribute
(SOAP 1.2 only)

Returns the URI of the SOAP node that generated this SOAP-fault object, if available. If the node information was not provided (which is always the case with a SOAP Version 1.1 connection), the attribute returns an empty string ("").

Data type: CHARACTER

Access: Read-only

Applies to: SOAP-fault object handle

Note: In SOAP Version 1.2, the combination of the SOAP-FAULT-NODE and SOAP-FAULT-ROLE attributes replaces the SOAP-FAULT-ACTOR attribute in SOAP Version 1.1. If the ABL client is connected via SOAP Version 1.1, the SOAP-FAULT-NODE attribute returns an empty string ("").

SOAP-FAULT-ROLE attribute
(SOAP 1.2 only)

Returns the URI that identifies the role in which the node that generated this SOAP-fault object was operating when the fault occurred. If the role information is not provided, the attribute returns an empty string (" ").

Data type: CHARACTER

Access: Read-only

Applies to: SOAP-fault object handle

Note: In SOAP Version 1.2, the combination of the SOAP-FAULT-NODE and SOAP-FAULT-ROLE attributes replaces the SOAP-FAULT-ACTOR attribute in SOAP Version 1.1. If the ABL client is connected via SOAP Version 1.1, the SOAP-FAULT-ROLE attribute returns an empty string ("").

SOAP-FAULT-STRING attribute

Returns the SOAP fault string describing the fault for this SOAP-fault object.

Data type: CHARACTER

Access: Read-only

Applies to: SOAP-fault object handle

SOAP-FAULT-SUBCODE attribute
(SOAP 1.2 only)

Returns the list of fully qualified sub-code names for this SOAP-fault object. If the subcode information is not provided, the attribute returns an empty string (" ").

Data type: CHARACTER

Access: Read-only

Applies to: SOAP-fault object handle

Note: If the ABL client is connected via SOAP Version 1.1, the SOAP-FAULT-SUBCODE attribute returns an empty string ("").

SOAP-VERSION attribute

Returns the SOAP version (for example "1.1", "1.2") of the Web service connection associated with a server handle. If the server handle does not have a Web service connection, the attribute returns the Unknown value (?)

Data type: CHARACTER

Access: Read-only

Applies to: Server object handle

SORT attribute

Indicates whether to sort new additions to the item list of a widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (column), COMBO-BOX widget, SELECTION-LIST widget

If the SORT attribute is TRUE, all items added to a combo box or selection list are added in sorted order. This means that the methods ADD-FIRST( ) and ADD-LAST( ) add items to the list in sorted order. The setting of this attribute has no affect on the function of the INSERT( ) and REPLACE( ) methods. Setting this attribute to FALSE returns these methods to their native function.

For browses, this attribute applies only to combo-box browse columns.

SORT-ASCENDING attribute

Indicates the sort order for a browse column.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (column)

When TRUE, the sort-arrow indicates that the column is sorted in ascending order. When FALSE, the sort-arrow indicates that the column is sorted in descending order. When the Unknown value (?), the column sort order is not indicated.

Note: Setting a sort-arrow indicator and sort-level number on a column does not change the column's sort order or the browse's query. The browse's query is responsible for actually sorting the data.

See also

SORT-NUMBER attribute

SORT-NUMBER attribute

Indicates the sort-level number for a browse column.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column)

Valid values are 1 through 9, inclusive, and the Unknown value (?). If the column is not sorted, returns the Unknown value (?).

Note: Setting a sort-arrow indicator and sort-level number on a column does not change the column's sort order or the browse's query. The browse's query is responsible for actually sorting the data.

See also

SORT-ASCENDING attribute

SSL-SERVER-NAME attribute

The name of the server for the current Secure Sockets Layer (TLS) session.

Data type: CHARACTER

Access: Read-only

Applies to: Server object handle, Socket object handle

For the application server or a socket object, this is the digital certificate subject name of the server for the current TLS session. It enables you to distinguish between the physical host name and the authenticated TLS server name.

For Web services, this is the digital certificate subject name of the `-SOAPEndpoint` (which is the URL identifying the endpoint for the Web service, not the server providing the WSDL).

When there is no socket connection, or the socket connection is not an TLS-based connection, the default value is the Unknown value (?).

STANDALONE attribute

Determines the value of the standalone string in the XML declaration of a SAX-writer object.

Data type: LOGICAL

Access: Readable/Writable

Applies to: SAX-writer object handle

You can set the attribute to the value of the standalone string in the XML declaration. The default value is the Unknown value (?). If the value is the Unknown value (?) then the standalone string will not appear in the XML declaration.

Valid values of `standalone` in the XML declaration are "yes" and "no". If you set the `standalone` value, then the standalone string appears in the XML declaration.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

This attribute can be read at all times, but can only be written when the WRITE-STATUS is either SAX-WRITE-IDLE or SAX-WRITE-COMPLETE. That is, it can only be changed when the writer is not writing, otherwise it will fail and generate an error message.

START-DOCUMENT( ) method

Creates the XML document with the prolog information.

Note: You must call the SET-OUTPUT-DESTINATION( ) method to set the output destination for the XML document before invoking this method.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
START-DOCUMENT( )
```

You must call this method to start the output before you call any other writing methods or none of the writing methods will succeed. After you call START-DOCUMENT, the WRITE-STATUS is changed to SAX-WRITE-BEGIN. If START-DOCUMENT is called while the SAX-writer is already writing (that is, with a status other than SAX-WRITE-IDLE or SAX-WRITE-COMPLETE), then the method fails.

If the SAX-writer is unable to write to the output destination, it generates an error message and change the WRITE-STATUS to SAX-WRITE-ERROR.

If the FRAGMENT attribute is FALSE, then the XML declaration is created. The version, encoding style, and standalone state of the document is specified in the declaration. If the ENCODING attribute is not set, the declaration defaults to UTF-8, but this value does not appear in the declaration. If the VERSION attribute is not set, it defaults to "1.0". If the STANDALONE attribute is not set, then the declaration default is to omit the attribute. Here is the default declaration:

```
<?xml version="1.0"?>
```

See also

END-DOCUMENT( ) method, SET-OUTPUT-DESTINATION( ) method

START-ELEMENT( ) method

Starts an XML node based upon the name of the node in a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
START-ELEMENT ( name [ , namespace-URI ] [ , SAX-attributes-handle ] )
```

name

A CHARACTER or LONGCHAR expression evaluating to the fully qualified or unqualified name of the element.

namespace-URI

A CHARACTER or LONGCHAR evaluating to the URI of the element, or an empty string ("") or the Unknown value (?) if the element doesn't contain a namespace. If the handle is the Unknown value (?), then no attributes are added. If the handle is not valid or not a SAX-attributes object, then the method fails and returns FALSE.

SAX-attributes-handle

A HANDLE to a SAX-attributes object. The attributes in the SAX-attributes object are added to the new XML element.

Starts an XML node. This method call changes the WRITE-STATUS to SAX-WRITE-TAG.

For every call of the START-ELEMENT( ) method, there must be a following corresponding call of the END-ELEMENT( ) method. All the parameter values from the two calls must match for the methods to correspond.

If *namespace-URI* is present, then the prefix will be resolved in the following order:

1. The method attempts to extract the namespace from the *name*.
2. The method attempts to extract the namespace from a previously declared namespace.
3. The method attempts to generate the default namespace.

If the name contains a prefix, *namespace-URI* is present, and this is the first instance of the *namespace-URI,* then the namespace is added to the element. This technique is equivalent to calling the DECLARE-NAMESPACE( ) method.

If only the *name* is present and it contains a prefix, then the SAX-writer attempts to resolve the prefix to a namespace.

If the *SAX-attributes-handle* is supplied, then attributes are written out after the name of the element, in the order that they are indexed in the SAX-attributes object. If they contain namespace information, then that information is handled as if the attribute were added using the INSERT-ATTRIBUTE( ) method.

If the STRICT attribute is TRUE, the FRAGMENT attribute is FALSE, and the call would result in more than one document-level element (that is, root node), then the method fails. Also, if STRICT is TRUE, an external DTD has been declared, and the call would create the root node, then the name used for the DTD declaration must match the name of the root node or the method fails.

See also

END-ELEMENT( ) method

STARTUP-PARAMETERS attribute

Returns a character string containing a comma-separated list of all startup parameters you defined at startup for the current ABL session.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

This list includes startup parameters defined in the ABL default startup parameter file (`$DLC/startup.pf`) or the file specified by the `$PROSTARTUP` environment variable, as well as startup parameters you specify on the command line or within a parameter file (`.pf`). The value of this attribute does not change during run time.

The startup parameter values in this list reflect initial parameter value settings. The value of a startup parameter during an ABL session can be different from its initial value setting. Some startup parameters have an equivalent session attribute you can use in ABL code to override the parameter value during a session. If you override a parameter value using an equivalent session attribute, the new value is not reflected in the list of startup parameters returned in this attribute.

If you defined any startup parameters in the default parameter file (`startup.pf`), or another parameter file specified by the Parameter File (`-pf`) startup parameter, the list includes the `-pf` *`filename`* parameter and all parameters defined in that parameter file, followed by `(end .pf)`. For example:

```
-pf dbconnect.pf,-db sports2020,-H pclsmith,-S 5000,(end .pf)
```

A parameter file appears in the list whether or not it contains startup parameters. If a parameter file does not contain startup parameters, it appears in the list in the following format:

```
-pf filename,(end .pf)
```

The default parameter file (`startup.pf`) always appears in the list. The AVM expands the filename of only the default parameter file. All other filenames appear in the list as specified.

Individual startup parameters, defined within a parameter file or on the command line, appear in the list in the following format:

```
-parameter-name[parameter-value][,-parameter-name[parameter-value]]...
```

If the startup parameter has no value, the list contains the startup parameter followed by a comma. No space appears before or after a comma, and no comma appears at the end of the list.

If the list of startup parameters includes duplicates, the last occurrence takes precedence and all other instances are ignored (even though they appear in the list).

If the list of startup parameters includes the Password (`-P`) or Proxy Password (`-proxyPassword`) parameters, the AVM substitutes six asterisks in place of the password value.

If the list of startup parameters contains a hyphen with no parameter name, the hyphen is ignored.

You can use the ENTRY function to parse the list of startup parameters. If you use the ENTRY function with the default delimiter (comma), the function separates the parameter entries wherever a comma appears. If a comma appears in the list as part of a parameter value, the function might not parse the list correctly. A comma separating two startup parameters, as opposed to being part of a parameter value, is always followed by `"-"` or `"(end .pf)"`. Based on this convention, you can examine the character(s) after a comma to determine whether the comma is separating two startup parameters or is part of a parameter value.

If you started your ABL session with the Statistics (`-y`), Statistics with **CTRL+C** (`-yc`), or Segment Statistics (`-yd`) startup parameter, you can use the SHOW-STATS statement to see the value of the STARTUP-PARAMETERS attribute. This statement includes the value of this attribute in the output to the `client.mon` file.

The following table shows examples of original command lines and their equivalent STARTUP-PARAMETERS attribute values.

Table 107: STARTUP-PARAMETERS attribute usage examples

Original command	Value of the STARTUP-PARAMETERS attribute
`prowin32 -db sports2020 -T c:\temp -H pclsmith -S 5000` Where `startup.pf` contains no startup parameters	`-pf c:\dlc\startup.pf,(end .pf),-db sports2020,-T c:\temp,-H pclsmith,-S 5000`

`prowin32 -T c:\temp`
 	
Where `startup.pf` contains:
 	
`-db sports2020`
`-H pclsmith`
`-S 5000`
	`-pf c:\dlc\startup.pf,-db sports2020,-H pclsmith,-S 5000,(end .pf),-T c:\temp`

`prowin32 -pf dbconnect.pf -T c:\temp`
 	
Where `startup.pf` contains no startup parameters and `dbconnect.pf` contains:
 	
`-db sports2020`
`-H pclsmith`
`-S 5000`
	`-pf c:\dlc\startup.pf,(end .pf),-pf dbconnect.pf,-db sports2020,-H pclsmith,-S 5000,(end .pf),-T c:\temp`

`prowin32 - -T d:\work100a -db mystore -1 -db corporate -H corpmachine -S 5000`
 	
Where `startup.pf` contains:
 	
`-T c:\temp`
	`-pf c:\dlc\startup.pf,-T c:\temp,(end .pf),-T d:\work100a,-db mystore,-1,-db corporate,-H corpmachine,-S 5000`

`prowin32 -U lsmith -P mypassword`
 	
Where `PROSTARTUP=c:\commonarea\dbconnect.pf` and `c:\commonarea\dbconnect.pf` contains:
 	
`-db sports2020`
`-H pclsmith`
`-S 5000`
	`-pf c:\commonarea\db.pf,-db sports2020,-H pclsmith,-S 5000,(end .pf),-U lsmith,-P ******`

STATE-DETAIL attribute

A description that provides detail about the current state of the client-principal object.

Data type: CHARACTER

Access: Read-only

Applies to: Client-principal object handle

The AVM sets the value of this attribute, along with the LOGIN-STATE attribute, whenever the state of a client-principal object changes. You can also set the value of this attribute when calling the AUTHENTICATION-FAILED( ) method to place a client-principal object in an authentication failed state in order to specify a reason for the authentication failure.

See also

AUTHENTICATION-FAILED( ) method, LOGIN-STATE attribute

STATISTICS attribute

Enables the collection of statistics from the profiler. When set to TRUE, the profiler collects internal statics information about an ABL application. This information is in addition to the COVERAGE details. The data collected includes the number of internal procedures, external procedures, class methods, user-defined functions, temp-tables, static queries, static buffers, static datasets, r-code size, startup parameters, database connections, and tables per database during the session. The default value is FALSE.

Data type: LOGICAL

Access: Readable/Writable

Applies to: PROFILER system handle

STATUS-AREA attribute

Indicates whether a window has a status area.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WINDOW widget

If the STATUS-AREA attribute is TRUE, the window has a status area.

You can set this attribute only before the window is realized.

STATUS-AREA-FONT attribute
(Graphical interfaces only)

The font number of the font used in the status area of a window.

Data type: INTEGER

Access: Readable/Writeable

Applies to: WINDOW widget

The font number represents an entry in the font table maintained by the FONT-TABLE handle.

STOP attribute

Set to TRUE, if the asynchronous request was executing when the client issued the CANCEL REQUESTS( ) method.

Data type: LOGICAL

Access: Read-only

Applies to: Asynchronous request object handle

If the COMPLETE attribute is FALSE, the value of this attribute is the Unknown value (?). When the PROCEDURE-COMPLETE event is processed, this attribute is set to TRUE before the event procedure or method is executed if the remote request returned with an unhandled STOP condition; otherwise, it is set to FALSE.

STOP-OBJECT attribute

An object reference to an instance of a `Progress.Lang.Stop` class or one of its built-in subclasses. The instance describes a STOP condition raised by an asynchronous call to a procedure or Web service operation. This attribute holds the Unknown value (?) if there is no relevant STOP condition.

Data type: Progress.Lang.Stop class or a subclass

Access: Read-only

Applies to: Asynchronous request object handle

Notes

- Stop objects can be thrown by the server:
  - From a stop CATCH block
  - Whenever the server does not handle the corresponding STOP condition and the client has a relevant CATCH block on its portion of the call stack.

See also

ERROR-OBJECT attribute, Progress.Lang.Stop class

STOP-PARSING( ) method

Causes the parser to stop parsing the XML document. This lets an application search for particular data, then abort the parse as soon as the data are found.

Return type: LOGICAL

Applies to: SAX-reader object handle

Syntax

```
STOP-PARSING ( )
```

STOP-PARSING( ) can stop a parse started by SAX-PARSE( ), SAX-PARSE-FIRST( ) or SAX-PARSE-NEXT( ). That is, the parse can be single call or multiple scan.

STOP-PARSING sets the PARSE-STATUS attribute to SAX-COMPLETE.

Within a callback, to invoke STOP-PARSING( ), use the SELF system handle, as shown in the following fragment:

```
SELF:STOP-PARSING( ).
```

If STOP-PARSING( ) is invoked in a callback or in any procedure called directly or indirectly by a callback, the AVM continues to execute the callback as usual, but when the callback finishes, control returns to the next ABL statement after the most-recently-executed SAX-PARSE( ), SAX-PARSE-FIRST( ), or SAX-PARSE-NEXT( ).

STOPPED attribute

Indicates whether the last compilation stopped prior to completion.

Data type: LOGICAL

Access: Read-only

Applies to: COMPILER system handle

If set to TRUE, the `STOPPED` attribute indicates that the user interrupted the compile with **CTRL+C** or **CTRL+BREAK**. Because this raises the `STOP` condition you must handle it with an `ON STOP` block or `CATCH` block to look at the attribute.

STREAM attribute

A value that specifies the character set used for operating system file I/O — `"ibm850"` or `"iso8859-1"`.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

The Stream Character Set (`-stream`) parameter sets the value of this attribute.

This attribute is obsolete. See the CPSTREAM attribute.

STRETCH-TO-FIT attribute

Forces the image to expand or contract to fit within the image widget's boundaries.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: IMAGE widget

This attribute has no effect if an icon is displayed on the image widget.

STRICT attribute

Determines if the SAX-writer object should ensure that the XML document is well formed XML.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SAX-writer object handle

The default value, TRUE, directs the object to ensure that the output is well formed XML. If a method call results in invalid XML, then the method fails, WRITE-STATUS is changed to SAX-WRITE-ERROR, and the stream is closed.

FALSE directs the object to generate warning messages and proceed with the write. If any warning message is generated, then the XML document will not be valid.

You can read this attribute at all times, but you can only write to it when the object's WRITE-STATUS is either SAX-WRITE-IDLE or SAX-WRITE-COMPLETE. That is, it can only be changed when the writer is not writing, otherwise it fails and generates an error message.

STRICT-ENTITY-RESOLUTION attribute

Indicates whether or not the XML parser (DOM or SAX) will attempt to resolve an external entity if the entity is located outside of the directories in the SCHEMA-PATH attribute of the given SAX-reader or X-document object handle.

Data type: LOGICAL

Access: Readable/Writable

Applies to: SAX-reader object handle, X-document object handle

If STRICT-ENTITY-RESOLUTION is TRUE, the parser will NOT attempt to resolve an external entity that is located outside of the directories in the SCHEMA-PATH attribute of the given SAX-reader or X-document object handle. If STRICT-ENTITY-RESOLUTION is FALSE, the parser will attempt to resolve an external entity. Setting this attribute to TRUE can help prevent an external caller from gaining access to data located in directories not included in the SCHEMA-PATH attribute.

You can set this behavior globally by using the `-strictEntityResolution` startup parameter. The default value of the attribute is set by the startup parameter, but you can change the attribute as needed to override the behavior set by the startup parameter. See the *Startup Command and Parameter Reference* for more information.

STRING-VALUE( ) method

The string value (which the AVM computes at run time) of the contents of the buffer-field object.

The STRING-VALUE( ) method uses the format attribute to convert the buffer value to a string.

Data type: CHARACTER

Access: Read-only

Applies to: Buffer-field object handle

Syntax

```
STRING-VALUE ( [i] )
```

i

An integer expression representing a subscript, for fields that have extents.

SUBTYPE attribute

The subtype of a widget.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget (column), COMBO-BOX widget, FILL-IN widget, MENU-ITEM widget, Server object handle, X-document object handle, X-noderef object handle

This attribute is read-only for combo boxes, the server object handle, the X-document object handle, and the X-noderef object handle. You can set this attribute only before the widget is realized.

For menu items, the value of this attribute is either `"NORMAL"`, `"SKIP"`, or `"RULE"`. `"NORMAL"` is the default—a menu item that can be chosen, a toggle-box item, or read-only text. (In this case the READ-ONLY and TOGGLE-BOX attributes determine the specific type of the menu item.) `"SKIP"` specifies a blank line in the menu. `"RULE"` specifies a visible horizontal line in the menu.

For combo boxes, the value of this attribute is either `"SIMPLE"`, `"DROP-DOWN"`, or `"DROP-DOWN-LIST"`. The DROP-DOWN-LIST subtype is the default. The SIMPLE and DROP-DOWN subtypes apply only to character-field or character-variable combo-box widgets in graphical interfaces only, and only in Windows. If you set the subtype of a combo-box widget to `"SIMPLE"` or `"DROP-DOWN"` in a character interface, the AVM treats the combo-box widget as having the `"DROP-DOWN-LIST"` subtype.

For browses, this attribute applies only to combo-box browse columns.

For fill-ins, the value of this attribute is either `"PROGRESS"` or `"NATIVE"`. `"PROGRESS"` is the default. If set to `"PROGRESS"`, the fill-in widget has the behavior of a standard ABL field in character mode. Otherwise, the field has the behavior of a fill-in that is native to the current graphical environment. The NATIVE option of the VIEW-AS phrase specifies that the field adhere to the native behavior of the current window system or environment.

For the X-document object handle or X-noderef object handle, this attribute returns the name of the object type (character representation of the DOM NodeType), which will be one of the following: `"ATTRIBUTE"`, `"CDATA-SECTION"`, `"COMMENT"`, `"DOCUMENT"`, `"DOCUMENT-FRAGMENT"`, `"ELEMENT"`, `"ENTITY-REFERENCE"`, `"PROCESSING-INSTRUCTION"`, or `"TEXT"`.

For the server object handle, this attribute identifies the type of server to which the server object is bound. This is either an application server or a Web service. This attribute is set during the execution of the CONNECT( ) method and can be one of three values. Before you invoke the CONNECT( ) method, the attribute value is set to the Unknown value (?). Once you invoke the CONNECT( ) method, the attribute value is set to either `"APPSERVER"` for an application server or `"WEBSERVICE"` for a Web service.

SUPER-PROCEDURES attribute

A list of the super procedure handles associated with a procedure file or with the current ABL session. The handles appear in last in first out (LIFO) order, comma-delimited, in character format. Returns the empty string for a Web service procedure.

Data type: CHARACTER

Access: Read-only

Applies to: Procedure object handle, SESSION system handle, SOURCE-PROCEDURE system handle, TARGET-PROCEDURE system handle, THIS-PROCEDURE system handle

If there are no super procedures associated with a procedure file or with the current ABL session, the value of the SUPER-PROCEDURES attribute is the empty string.

SUPPRESS-NAMESPACE-PROCESSING attribute

Indicates whether namespace processing is suppressed.

Data type: LOGICAL

Access: Readable/Writable

Applies to: SAX-reader object handle, X-document object handle

FALSE, the default, indicates that namespace processing is not suppressed. TRUE indicates that namespace processing is suppressed.

For more information on accessing XML documents using the Document Object Model (DOM) and Simple API for XML (SAX) interfaces, see *Use XML with ABL Applications*.

SUPPRESS-WARNINGS attribute

Indicates whether the AVM suppresses warning messages during the session.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

If TRUE, the AVM does not display warning messages during the session. The default value of this attribute is FALSE unless the Suppress Warnings (`-sw`) startup parameter is used, in which case the attribute defaults to TRUE. For more information on the `-sw` startup parameter, see *Startup Command and Parameter Reference*.

SUPPRESS-WARNINGS-LIST attribute

A character expression with a comma separated list of warning message numbers to suppress.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SESSION system handle

If the Suppress Warning List (`-swl`) startup parameter is used, the attribute is set to the list specified by the startup parameter. If the Suppress Warnings (`-sw`) startup parameter is used or the SUPPRESS-WARNINGS attribute is set to TRUE, the list in the SUPPRESS-WARNINGS-LIST attribute is ignored and all warnings are suppressed. For more information on the `-sw` and `-swl` startup parameters, see *Startup Command and Parameter Reference*.

SYMMETRIC-ENCRYPTION-ALGORITHM attribute

A character string that specifies the name of the default cryptographic algorithm to use with the ENCRYPT and DECRYPT functions. The default value is `"AES_CBC_128"`.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SECURITY-POLICY system handle

This string is a concatenation of three character expressions that identify an algorithm, mode, and key size. For a list the supported cryptographic algorithms, see the SYMMETRIC-SUPPORT attribute reference entry.

You are responsible for generating, storing, and transporting this value.

See also

DECRYPT function, ENCRYPT function

SYMMETRIC-ENCRYPTION-IV attribute

The default initialization vector value to use with the encryption key in the ENCRYPT and DECRYPT functions. The default value is the Unknown value (?), which indicates that no initialization vector value is used.

Data type: RAW

Access: Readable/Writeable

Applies to: SECURITY-POLICY system handle

Using an initialization vector value increases the strength of the specified encryption key (that is, it makes the key more unpredictable).

You are responsible for generating, storing, and transporting this value.

See also

DECRYPT function, ENCRYPT function

SYMMETRIC-ENCRYPTION-KEY attribute

The default encryption key (a binary value) to use with the ENCRYPT and DECRYPT functions. The default value is the Unknown value (?).

Data type: RAW

Access: Write-only

Applies to: SECURITY-POLICY system handle

You may specify this key as a MEMPTR, CHARACTER, or LONGCHAR value, but the AVM treats it as a RAW.

If the value of this attribute is the Unknown value (?), you must provide the encryption key as an argument to the ENCRYPT and DECRYPT functions.

The AVM compares the size of the specified encryption key to the key size specified by the cryptographic algorithm. If the key sizes are inconsistent, the AVM generates a run-time error.

The AVM obscures this attribute value to protect it against unauthorized access. You are responsible for generating, storing, and transporting this value.

You can generate an encryption key, based on the PKCS#5/RFC 2898 standard, by using either the GENERATE-PBE-KEY function or the GENERATE-RANDOM-KEY function.

Note: Do not use the GENERATE-RANDOM-KEY function to assign a key value to this attribute directly. Doing so will render the key irretrievable (as this attribute is write-only).

See also

DECRYPT function, ENCRYPT function

SYMMETRIC-SUPPORT attribute

Returns a comma-separated list of supported cryptographic algorithm names to use in encrypting and decrypting data. Each algorithm name is a concatenation of three character expressions that identify an algorithm, mode, and key size.

Data type: CHARACTER

Access: Read-only

Applies to: SECURITY-POLICY system handle

The following table lists the supported cryptographic algorithm names.

Table 108: Supported cryptographic algorithm names

AES_CBC_128	AES_OFB_256
AES_CBC_192	DES_CBC_56
AES_CBC_256	DES_CFB_56
AES_CFB_128	DES_ECB_56
AES_CFB_192	DES_OFB_56
AES_CFB_256	DES3_CBC_168
AES_ECB_128	DES3_CFB_168
AES_ECB_192	DES3_ECB_168
AES_ECB_256	DES3_OFB_168

AES_OFB_128	RC4_ECB_128
AES_OFB_192	–

See also

DECRYPT function, ENCRYPT function

SYNCHRONIZE( ) method

Synchronizes a hierarchy of data-relation queries on a parent buffer.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
hBuff:SYNCHRONIZE ( )
```

hBuff

The handle to any parent buffer from among the available temp-table buffers of a ProDataSet.

This method traverses the ProDataSet object hierarchy starting at buffer *hBuff* and reopens each data-relation query for the current parent at each lower level. Use this method to populate one or more related child buffers for the ProDataSet object buffer.

By default, if the query is associated with a browse, the synchronize action automatically refreshes the browse. If the query is not associated with a browse, the synchronize action automatically gets the first buffer in the query by invoking a GET FIRST operation. If there is a REPOSITION data relation and no browse, the synchronize action gets the next record in the query by invoking a GET NEXT operation.

SYSTEM-ALERT-BOXES attribute

Indicates whether the AVM displays system messages in alert boxes.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

If TRUE, the AVM displays system messages in alert boxes rather than in the message area.

SYSTEM-ID attribute

Returns the system ID of the external DTD from which an XML document was generated. This contains the path to the DTD which is either a file system path or an HTTP URL. The ABL parser uses this information to retrieve the DTD when parsing the document.

Data type: CHARACTER

Access: Read-only

Applies to: X-document object handle

TAB-POSITION attribute

The tab order of a widget within its field group.

Data type: INTEGER

Access: Read-only

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

You can change the tab order of the widget at the field level using the MOVE-BEFORE-TAB-ITEM( ) or MOVE-AFTER-TAB-ITEM( ) methods, and at the field group level using the FIRST-TAB-ITEM attribute or LAST-TAB-ITEM attribute.

TAB-STOP attribute

Returns TRUE if the widget is in its parent's tab chain.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

Setting the TAB-STOP attribute to FALSE removes the widget from its parent's tab chain. Setting the TAB-STOP attribute to TRUE adds the widget to the end of its parent's tab chain. If the widget is already in the tab chain, its position does not change.

In Windows, the mnemonic key (**ALT** accelerator) for a widget will not work if the widget is removed from the tab order. Also, because the widget is not in the tab order, pressing **TAB** will not change focus from the widget.

TABLE attribute

The name of the database table or temp-table containing the field associated with a widget, buffer, or buffer-field.

Data type: CHARACTER

Access: Read-only

Applies to: BROWSE widget (browse and column), Buffer object handle, Buffer-field object handle, COMBO-BOX widget, EDITOR widget, FILL-IN widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

Note: The TABLE attribute of a buffer contains the name of the table, not the name of the buffer.

TABLE-CRC-LIST attribute

Returns a comma-separated list of the CRC value for each table referenced in the r-code file specified by the RCODE-INFO:FILE-NAME attribute.

Data type: CHARACTER

Access: Read-only

Applies to: RCODE-INFO handle

This list corresponds directly to the list generated by the TABLE-LIST attribute.

Use this attribute with the TABLE-LIST attribute to compare the CRC value for all tables referenced in the r-code file with those stored in the database to determine whether or not a procedure file needs to be recompiled after a database change.

If the r-code file was compiled without table references, this attribute returns the empty string ("").

TABLE-HANDLE attribute

Returns the handle of a temp-table object, if any, associated with the buffer object. If the buffer is not associated with a temp-table object, it returns the Unknown value (?).

Data type: HANDLE

Access: Read-only

Applies to: Buffer object handle

This attribute allows you to delete a default buffer object for a temp-table object by deleting the temp-table object (since it is illegal to delete the default buffer object itself).

TABLE-LIST attribute

Returns a comma-separated list of all tables referenced in the r-code file specified by the `RCODE-INFO:FILE-NAME` attribute. Each table reference includes the table name and related database name (for example: `Sports2020.CUSTOMER`).

Data type: CHARACTER or LONGCHAR

Access: Read-only

Applies to: RCODE-INFO handle

This list corresponds directly to the list generated by the TABLE-CRC-LIST attribute.

Use this attribute with the TABLE-CRC-LIST attribute to compare the CRC value for all tables referenced in the r-code file with those stored in the database to determine whether or not a procedure file needs to be recompiled after a database change.

If the r-code file was compiled without table references, this attribute returns the empty string ("").

Note: The TABLE-LIST attribute returns a CHARACTER value unless the list of tables exceeds 32K. If it exceeds 32K, TABLE-LIST returns a LONGCHAR value.

TABLE-NUMBER attribute

The sequence number, within the database, of the table that corresponds to a buffer.

Data type: INTEGER

Access: Read-only

Applies to: Buffer object handle

Tag property (ActiveX control)
(Windows only; Graphical interfaces only)

A variable that lets the developer store an arbitrary string value.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Any ActiveX control

The Tag property is an extended ActiveX control property that lets the developer store an arbitrary string value and retrieve it later. The AVM does not use this property internally; rather, the property lets the developer store application-specific information with the control.

This property is initialized to an empty string.

Note: The length of the string cannot exceed 2,147,483, 647 characters.

TEMP-DIRECTORY attribute

The name of the directory in which the AVM stores temporary files during the session.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

By default, this is the current working directory. Otherwise, it is the directory specified using the Temporary Directory (-T) parameter.

TEMP-TABLE-PREPARE( ) method

Signifies that all the field and index definitions for a temp-table have been supplied.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
TEMP-TABLE-PREPARE ( temp-table-name-exp [ , before-table-exp] )
```

temp-table-name-exp

A CHARACTER expression that evaluates to a temp-table name to be used in subsequent query statements that refer to this temp-table.

before-table-exp

A LOGICAL expression that allows you to explicitly generate a before-table for a dynamic temp-table. TRUE indicates the AVM will create the before-table. (The AVM generates the name for this before-table.) The default value is FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

Note: When passing a temp-table to a method (or procedure/function), you are required to have compatible schema for the table on both the caller and called sides. If the temp-table on either side has a before-table, in some cases the other side may also be required have a before-table, or the AVM will raise an error.

The temp-table is in an UNPREPARED state after the first definitional method is called until this method is called. During this time, only ADD/CREATE type methods may be called.

The TEMP-TABLE-PREPARE( ) method must be called after all fields and indexes have been created and before any non-ADD/CREATE method can be called. This method causes the pending list of field and index definitions to become part of the actual temp-table object, which puts the temp-table in a PREPARED state (that is, makes it ready for use).

Note: The NAME attribute of the Temp-table object handle is writeable for dynamic and AVM-generated temp-tables. A new temp-table name cannot be assigned until the TEMP-TABLE-PREPARE( ) method has been executed.

TENANT-ID( ) method

Returns the tenant ID that is recorded in the client-principal object for an OpenEdge multi-tenant database. A tenant ID is returned if this client-principal instance has been used to set the connection identity for the specified multi-tenant database. The association of this tenant ID and multi-tenant database persists for this instance of the client-principal and cannot be removed for the duration of the user login session identified by this client-principal.

Return type: INTEGER

Applies to: Client-principal object handle

Syntax

```
TENANT-ID ( [ db-exp ] )
```

db-exp

An optional character expression that evaluates to a case-insensitive logical or alias name of a multi-tenant OpenEdge RDBMS. This expression can be unspecified or evaluate to the Unknown value (?) only if the client-principal object has been used to establish identity for a single OpenEdge database connection.

The method can return the following values:

- **A recorded tenant ID** — If tenant information is stored in the client-principal for the database specified by *db-exp*
- **The Unknown value (?)** —
  - If no tenant information is stored in the client-principal for the database specified by *db-exp*, or
  - If the specified database has multi-tenancy enabled and the client-principal object has not been used to set its connection identity, or
  - If the client-principal object is generated by an external security system
- **Zero (0)** — If the database specified by *db-exp* does not have multi-tenancy enabled

Note: Zero (0) can also be the ID of a valid tenant in a database configured for multi-tenancy. Therefore, the tenant ID return value is not a reliable indicator if a database has multi-tenancy enabled.

ABL raises ERROR if:

- The client-principal object is in the INITIAL or FAILED state (see the LOGIN-STATE attribute entry)
- The expression specified by *db-exp* evaluates to the Unknown value (?) and multiple OpenEdge databases are connected

See also

DB-LIST attribute, TENANT-NAME( ) method

TENANT-NAME( ) method

Returns the tenant name that is recorded in the client-principal object for an OpenEdge multi-tenant database. A tenant name is returned if this client-principal instance has been used to set the connection identity for the specified multi-tenant database. The association of this tenant name and multi-tenant database persists for this instance of the client-principal and cannot be removed for the duration of the user login session identified by this client-principal.

Return type: CHARACTER

Applies to: Client-principal object handle

Syntax

```
TENANT-NAME ( [ db-exp ] )
```

db-exp

An optional character expression that evaluates to a case-insensitive logical or alias name of a multi-tenant OpenEdge RDBMS. This expression can be unspecified or evaluate to the Unknown value (?) only if the client-principal object has been used to establish identity for a single OpenEdge database connection.

The method can return the following values:

- **A recorded tenant name** — If tenant information is stored in the client-principal for the database specified by *db-exp*
- **The Unknown value (?)** —
  - If no tenant information is stored in the client-principal for the database specified by *db-exp*, or
  - If the client-principal object is generated by an external security system
- **Blank ("")** — If the database specified by *db-exp* does not have multi-tenancy enabled or the specified database has multi-tenancy enabled, but the client-principal object has not been used to set its connection identity

ABL raises ERROR if:

- The client-principal object is in the INITIAL or FAILED state (see the LOGIN-STATE attribute entry)
- The expression specified by *db-exp* evaluates to the Unknown value (?) and multiple OpenEdge databases are connected

See also

DB-LIST attribute, TENANT-ID( ) method

TEXT-SELECTED attribute

Indicates whether text is currently selected in a widget.

Data type: LOGICAL

Access: Read-only

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

The TEXT-SELECTED attribute is TRUE if text in the widget is currently selected.

THREAD-SAFE attribute

Indicates whether the DLL or shared library call specified is thread safe.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Call object handle

If TRUE, the specified DLL or shared library call is thread safe. When a DLL or shared library has been marked as THREAD-SAFE, multiple sessions can access it simultaneously. The value of this attribute is ignored if the LIBRARY attribute is not set or if the application is not running on an instance of the Progress Application Server for OpenEdge.

THREE-D attribute
(Windows only; Graphical interfaces only)

Indicates whether the AVM displays widgets using a three-dimensional format.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget, SESSION system handle, WINDOW widget

If TRUE, the specified widgets are displayed in a three-dimensional format. For a frame or dialog box, any field-level widgets in the frame or dialog box are also displayed in three-dimensional format. If a frame has the THREE-D attribute set to TRUE, the default background color is the color Button Face rather than the color Window. For a window, setting this attribute changes the window background color to color Button Face only, and has no effect on any widgets contained in the window. Frames do not inherit the THREE-D attribute from a window or ancestor frame.

If the THREE-D attribute is TRUE for the SESSION handle, then all system dialog boxes and alert boxes are displayed in three-dimensional format.

You can set this attribute only before the widget is realized.

Note: To maintain size compatibility, the AVM sets the default vertical size of two-dimensional fill-ins equal to the vertical size of three-dimensional fill-ins. Also, ABL does not fully support the overlay of three-dimensional widgets. For more information, see the section on three-dimensional layout in *OpenEdge Programming Interfaces*.

TIC-MARKS attribute
(Windows only; Graphical interfaces only)

Enables the display of short hash marks on the outside of a slider to help indicate the movement of the trackbar with the slider widget. The default is not to display tic marks. If you specify the TIC-MARKS option, it is assumed that you are using new code to create a slider, and the trackbar on the slider widget will be relatively large.

However, if you omit the TIC-MARKS option, the AVM assumes that you are migrating old code, and the default size of the slider is the size originally defined for the slider in the old code.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SLIDER widget

If you want to use the large trackbar but do not want tic marks to display, specify TIC-MARKS NONE.

To implement the TIC-MARKS option, you must also specify on which side, or sides, of the trackbar tic-marks appear by using the additional qualifying values. The following table lists and defines these values.

Table 109: TIC-MARK values

Value	Description
TOP	TIC-MARKS appear on the top of the slider only
BOTTOM	TIC-MARKS appear on the bottom of the slider only
LEFT	TIC-MARKS appear on the left side of the slider only
RIGHT	TIC-MARKS appear on the right side of the slider only
BOTH	TIC-MARKS appear on both sides of the slider

The TIC-MARKS attribute must be set before the slider is realized. Also, you can use the FREQUENCY attribute with the TIC-MARKS attribute to indicate how frequently a tic mark will display along the trackbar of a slider.

TIME-SOURCE attribute

Specifies the client or database server machine that serves as the time source for applications running during the ABL session.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SESSION system handle

TIME-SOURCE accepts either of the following settings:

- "`local`" or the null string ("") — Your application uses the client machine as its time source. The default value is "`local`".
- "*dbname*" — Your application uses the machine running the server for the database with the name *dbname* as its time source.

All language elements that involve the current time, such as the MTIME, NOW, TIME, TIMEZONE, and TODAY functions, use the specified time source. This attribute is useful for client/server applications that span time zones.

Notes

- If the TIME-SOURCE attribute is set to a value other than "`local`" or the null string (""), then the SESSION:TIMEZONE attribute is ignored for the functions described above.

TIMEZONE attribute

Specifies the time zone offset from UTC, in minutes, to use for the current session.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SESSION system handle

The AVM uses the TIMEZONE value to calculate the results of:

- the functions DATETIME-TZ, MTIME, NOW, TIME, TIMEZONE, and TODAY
- direct assignment operations between DATE, DATETIME, and DATETIME-TZ data types
- the special TODAY and NOW initial values for database and temp-table fields with DATE, DATETIME, and DATETIME-TZ data types.

For example, the following code shows how TIMEZONE Is used in a direct assignment of a DATE value to a DATETIME-TZ value:

```
DEFINE VARIABLE dDate AS DATE NO-UNDO.
DEFINE VARIABLE tzDate AS DATETIME-TZ NO-UNDO.

dDate = 1/25.
tzDate = dDate.
DISPLAY tzDate.
```

Because the DATETIME-TZ data type contains more information than a DATE data type, the time value of `tzDate` defaults to midnight. If SESSION:TIMEZONE is not set, the timezone defaults to the session's timezone, and the code displays:

```
01/25/2016 00:00:00.000-5:00
```

However, if SESSION:TIMEZONE is set to 120, the timezone is offset from UTC by two hours, and the code displays:

```
01/25/2016 07:00:00.000+2:00
```

For more information on assignments with time data types, see the ASSIGN statement.

TITLE attribute

The title string a widget displays.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget, DIALOG-BOX widget, FRAME widget, MENU widget (pop-up only), WINDOW widget

For browse widgets, pop-up menus, and frames, this attribute is writeable only before the widget is realized. However, you can modify an existing frame title after realization.

TITLE-BGCOLOR attribute
(Graphical interfaces only)

The color number for the background color of the widget title.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget, DIALOG-BOX widget, FRAME widget, MENU widget (pop-up only)

The color number represents an entry in the color table maintained by the COLOR-TABLE handle. This attribute is read-only for all applicable widget types. It is writeable only for backward compatibility and has no effect when you set it.

TITLE-DCOLOR attribute
(Character interfaces only)

The color number for the character-mode display color of the widget title.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget, DIALOG-BOX widget, FRAME widget, MENU widget (pop-up only)

The color number represents an entry in the color table maintained by the COLOR-TABLE handle. This attribute is read-only for browse widgets.

TITLE-FGCOLOR attribute
(Graphical interfaces only)

The color number for the foreground color of the widget title.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget, DIALOG-BOX widget, FRAME widget, MENU widget (pop-up only)

The color number represents an entry in the color table maintained by the COLOR-TABLE handle. This attribute is read-only for all applicable widget types. It is writeable only for backward compatibility and has no effect when you set it.

TITLE-FONT attribute

The font number for the font of the widget title.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget, DIALOG-BOX widget, FRAME widget, MENU widget (pop-up only)

The font number represents an entry in the font table maintained by the FONT-TABLE handle. This attribute is read-only for all applicable widget types. It is writeable only for backward compatibility and has no effect when you set it.

TOGGLE-BOX attribute

Indicates whether a menu-item appears and acts like a toggle box.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: MENU-ITEM widget

If the TOGGLE-BOX attribute is TRUE, the menu item appears and interacts like a toggle box. You can set this attribute only before the widget is realized.

TOOLTIP attribute
(Windows only; Graphical interfaces only)

A help text message for a text field or text variable. The AVM automatically displays this text when the user pauses the mouse pointer over a widget for which a tooltip is defined.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, IMAGE widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, and TOGGLE-BOX widget

You can add or change the TOOLTIP attribute at any time. If TOOLTIP is set to "" or the Unknown value (?), then the ToolTip is removed. No ToolTip is the default.

TOOLTIPS attribute
(Windows only; Graphical interfaces only)

Indicates whether ToolTip information is displayed when the mouse pointer pauses over a control for which tooltip information is defined.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

If TRUE, the ToolTip information that is defined for any controls associated with a given session displays when the mouse pointer pauses over a control. Otherwise, ToolTip information does not display for any controls in the session.

Top property
(Windows only; Graphical interfaces only)

The vertical position of the control-frame and control-frame COM object from the top border of the parent container widget, in pixels.

Return type: INTEGER

Access: Readable/Writeable

Applies to: CONTROL-FRAME widget, COM object

Setting this value changes the ROW attribute and Y attribute of the corresponding control-frame widget to an equivalent value.

Note: References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the Accessing COM object properties and methods.

TOP-NAV-QUERY attribute

Lets you get or set the default navigation query for a top-level buffer in a ProDataSet object.

Note: A top-level buffer is a ProDataSet object buffer that is not a child in any active data relation. There may be one or more top-level buffers in a ProDataSet object.

Data type: HANDLE

Access: Readable/Writeable

Applies to: ProDataSet object handle

The syntax to read or set this attribute is as follows:

Syntax

```
TOP-NAV-QUERY ( index | buffer-name ) [ = query-object-handle ]
```

index

An integer expression that evaluates to the 1-based index of the top-level buffer.

buffer-name

A character expression that evaluates to the name of the top-level buffer.

query-object-handle

A handle to a query object.

See also

Query object handle

TOP-ONLY attribute

Indicates whether another frame or window can overlay a given frame or window, respectively.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget, WINDOW widget

If the TOP-ONLY attribute is TRUE for the frame, no other frame can overlay it.

If more than one window is designated as TOP-ONLY, they will all stay on top of all non-TOP-ONLY windows, but each can be brought to the foreground. That is, a TOP-ONLY window is always on top of all non-TOP-ONLY windows, but is not necessarily on top of all TOP-ONLY windows.

The TOP-ONLY behavior will be temporarily suspended while a dialog box is displayed to prevent the TOP-ONLY windows from covering the dialog-box.

A window cannot have both the TOP-ONLY and ALWAYS-ON-TOP attributes set to TRUE. Setting the TOP-ONLY attribute to TRUE will set the ALWAYS-ON-TOP attribute to FALSE. The default value of the TOP-ONLY attribute is FALSE.

When using .NET forms with ABL windows in an ABL session, setting this attribute to TRUE on an ABL window enables the specified window to overlay all other .NET forms and non-TOP-ONLY windows in the session. However, .NET forms do not have a corresponding property or method to specify similar behavior for a form. Thus, you cannot have a .NET form in an ABL session that overlays all other forms and windows in the session.

See also

ALWAYS-ON-TOP attribute

TRACE-FILTER attribute

Specifies a comma-separated list of procedure names for which the profiler records detailed timing information. Requires access to source code; not intended for use in production systems.

Data type: CHARACTER

Access: Readable/Writable

Applies to: PROFILER system handle

The initial value is an empty string (""). The expression patterns are interpreted in the same way as does the ABL built-in MATCHES() function. If a procedure name matches one of the patterns on the list, then the profiler records detailed timing information for every executable line in that procedure. The pattern matching is not case sensitive.

In addition to recording summary timing data for each executable line, TRACE-FILTER records timing information each time a statement is executed. With this detailed information, a complete execution trace for the application can be established.

If the value of TRACE-FILTER is " ", then the pattern does not match any procedure names. When no procedure names are matched, the only tracing information the profiler records is what is specified by the PROFILER:TRACING attribute. If the value is "*", then the pattern matches all procedure names.

Examples: To get tracing information for all enable_UI procedures, set TRACE-FILTER to "enable_UI *". To get tracing information for all enable_UI procedures and also all executable lines in the procedure hello.p, set TRACE-FILTER to "enable_UI*,*hello.p".

The TRACE-FILTER patterns are examined by the profiler only as it analyzes the data in the raw data temporary file, in preparation for writing data to the output file. This usually occurs during PROFILER:WRITE-DATA() calls or at the end of the session. In theory, the TRACE-FILTER value can be set any time before the profiler writes the data for the session to its output file. However, when the raw data temporary file becomes too large, the profiler does a partial analysis on the fly. To avoid this, set the TRACE-FILTER pattern before any procedure intended to match the pattern is run. Redundant entries in the PROFILER:TRACE-FILTER list are automatically removed.

See also

PROFILER:TRACING

TRACING attribute

Formats the output for TRACE-FILTER. Requires access to source code; not intended for use in production systems.

Data type: CHARACTER

Access: Readable/Writable

Applies to: PROFILER system handle

The initial value is an empty string (""). The value is a comma-separated list of procedure names and listing line numbers corresponding to specific ABL statements for which the profiler records detailed timing information. Each element in the list is *procedure-name* | *line-number* where *procedure-name* is the name of a `main.p` procedure (not an internal procedure) and *line-number* is the debug listing line number of the statement to be traced. The list elements are separated by commas.

If TRACING is set to an empty string (""), then the only tracing information that is recorded is what is specified by the PROFILER:TRACE-FILTER attribute. The TRACING list is examined by the profiler only as it is analyzing the data in the raw data temporary file in preparation for writing data to the output file. This examination usually occurs during PROFILER:WRITE-DATA() calls or at the end of the session. In theory, then the TRACING value can be set any time before the profiler writes the data for the session to its output file. However, when the raw data temporary file becomes too large, the profiler does a partial analysis on the fly. To avoid this, set the TRACING list before any statements to be traced are executed. Redundant entries in the PROFILER:TRACING are automatically removed.

See also

TRACE-FILTER attribute

TRACKING-CHANGES attribute

Set to TRUE to start tracking changes to the data in an individual ProDataSet temp-table. The AVM tracks changes to the temp-table until you set this attribute to FALSE. When this attribute is FALSE, any changes you make to the data in the temp-table are considered part of the fill process. The default value is FALSE.

For all other temp-tables that are not part of a ProDataSet, this attribute has the Unknown value (?).

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Temp-table object handle

When the TRACKING-CHANGES attribute is set to TRUE for a ProDataSet temp-table, the AVM tracks changes to the data in that temp-table using a before-image table that contains the original version of each row. You can think of the temp-table itself as the after-image because it contains the latest version of each row.

Note: If you SHARE a static ProDataSet, you must also SHARE all of its temp-tables. A SHARED static temp-table cannot have a before-image table. Therefore, you cannot track changes to its temp-tables.

Every row in the after-image table that has been modified or created corresponds to a row in the before-image table. Deleted rows do not appear in the after-image table, because it reflects the current state of the data. Every row in the before-image table has a non-zero ROW-STATE attribute value, because every row in the before-image table is the before-image of a deleted, created, or modified row in the after-image table. Unchanged rows do not appear in the before-image table.

You can track newly created rows and changed rows through either the before-image table or the after-image of the table. However, since deleted rows do not appear in the after-image table, it is better to track changes through the before-image table.

You might want to set TRACKING-CHANGES back to FALSE for a temp-table when you are ready to:

- Accept the changes using the ACCEPT-CHANGES( ) method or the ACCEPT-ROW-CHANGES( ) method
- Reject the changes using the REJECT-CHANGES( ) method or the REJECT-ROW-CHANGES( ) method
- Get and merge the changes using the GET-CHANGES( ) method and the MERGE-CHANGES( ) method or MERGE-ROW-CHANGES( ) method, respectively

Note: If you pass a ProDataSet remotely or locally by value and `TRACKING-CHANGES` is TRUE for the ProDataSet or any of its tables, the before- and after-image-tables are passed in their current state, but the value of the `TRACKING-CHANGES` attribute is not passed. That is, it is not automatically set to TRUE on the receiving side.

TRANSACTION attribute

A handle to the current transaction object. Returns the Unknown value (?) for a Web service procedure.

Data type: HANDLE

Access: Read-only

Applies to: Procedure object handle, SOURCE-PROCEDURE system handle, TARGET-PROCEDURE system handle, THIS-PROCEDURE system handle

The transaction handle returned by this attribute provides attributes and methods that allow you to manage a transaction object running on an application server. In an ABL client session, or in an application server session that has no active transaction initiating procedure, you can only use the IS-OPEN attribute to check whether a transaction is open.

For more information on transaction initiating procedures, see the TRANSACTION-MODE AUTOMATIC statement. For more information on the attributes and methods provided by the transaction handle, see the Transaction object handle.

TRANSPARENT attribute

Makes the background color of the image transparent. The background color is determined by the color of the pixel in the lower-left corner of the image.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: IMAGE widget

The TRANSPARENT attribute overrides the CONVERT-3D-COLORS attribute; if both TRANSPARENT and CONVERT-3D-COLORS are set to TRUE, CONVERT-3D-COLORS is ignored.

This attribute has no effect if an icon is displayed on the image widget.

TRANS-INIT-PROCEDURE attribute
(PAS for OE only)

The handle to the transaction initiating procedure that started the currently-open automatic transaction.

Data type: HANDLE

Access: Read-only

Applies to: Transaction object handle

You can use this procedure handle to access the attributes and methods of the active transaction initiating procedure or to delete the procedure, thus terminating the automatic transaction.

If no automatic transaction is active, TRANS-INIT-PROCEDURE returns an invalid handle. To check a handle for validity, use the VALID-HANDLE function.

For information on automatic transaction initiating procedures, see the TRANSACTION-MODE AUTOMATIC statement reference entry.

TYPE attribute

The type of a handle.

Data type: CHARACTER

Access: Read-only

Applies to: Asynchronous request object handle, AUDIT-CONTROL system handle, AUDIT-POLICY system handle, BROWSE widget (browse, column, and cell), Buffer object handle, Buffer-field object handle, BUTTON widget, Call object handle, Client-principal object handle, CLIPBOARD system handle, CODEBASE-LOCATOR system handle, COLOR-TABLE system handle, COMBO-BOX widget, COMPILER system handle, CONTROL-FRAME widget, CURRENT-WINDOW system handle, Data-relation object handle, Data-source object handle, DEBUGGER system handle, DEFAULT-WINDOW system handle, DIALOG-BOX widget, DSLOG-MANAGER system handle, EDITOR widget, ERROR-STATUS system handle, FIELD-GROUP widget, FILE-INFO system handle, FILL-IN widget, FOCUS system handle, FONT-TABLE system handle, FRAME widget, IMAGE widget, LAST-EVENT system handle, LITERAL widget, LOG-MANAGER system handle, MENU widget, MENU-ITEM widget, Procedure object handle, ProDataSet object handle, PROFILER system handle, Query object handle, RADIO-SET widget, RCODE-INFO handle, RECTANGLE widget, SAX-attributes object handle, SAX-reader object handle, SAX-writer object handle, SECURITY-POLICY system handle, SELECTION-LIST widget, SELF system handle, Server object handle, Server socket object handle, SESSION system handle, SLIDER widget, SOAP-fault object handle, SOAP-fault-detail object handle, SOAP-header object handle, SOAP-header-entryref object handle, Socket object handle, SOURCE-PROCEDURE system handle, Stream object handle, SUB-MENU widget, TARGET-PROCEDURE system handle, Temp-table object handle, TEXT widget, THIS-PROCEDURE system handle, TOGGLE-BOX widget, Transaction object handle, WEB-CONTEXT system handle, WINDOW widget, X-document object handle, X-noderef object handle

The TYPE attribute returns the widget or handle type. Some examples are `"WINDOW"`, `"FRAME"`, `"BUTTON"`, `"MENU"`, `"SAX-ATTRIBUTES"`, `"SAX-READER"`, and `"SERVER"`.

For application server and Web service handles, the TYPE attribute returns `"SERVER"`.

If a system handle (such as CURRENT-WINDOW or FOCUS) refers to a user interface widget, then the TYPE attribute returns the type of that widget. If a system handle (such as SESSION or CLIPBOARD) refers to an ABL status or system object, then the TYPE attribute value is `"PSEUDO-WIDGET"`.

For procedure handles and system handles that refer to procedures (such as THIS-PROCEDURE), the TYPE attribute returns "PROCEDURE".

For an asynchronous request handle, the TYPE attribute returns "ASYNC-REQUEST".

For the ProDataSet, data-relation, and data-source object handles, this attribute returns "DATASET", "DATA-RELATION", and "DATA-SOURCE", respectively.

For a server-socket handle, the TYPE attribute returns "SERVER-SOCKET", and for a socket handle, it returns "SOCKET".

For the SOAP-fault and SOAP-fault-detail object handles, this attribute returns "SOAP-FAULT" and "SOAP-FAULT-DETAIL", respectively.

For the SOAP-header and SOAP-header-entryref object handles, this attribute returns "SOAP-HEADER" and "SOAP-HEADER-ENTRYREF", respectively.

For the X-document and X-noderef object handles, the TYPE attribute returns "X-DOCUMENT" and "X-NODEREF", respectively.

For browse columns, the TYPE attribute returns "COMBO-BOX", "FILL-IN", or "TOGGLE-BOX". If you specify the Browse Column Type (-browcoltype) startup parameter, then the TYPE attribute for browse columns returns "BROWSE-COLUMN" regardless of the actual column type.

UNDO attribute

If TRUE, the temp-table is UNDO; if FALSE, the temp-table is NO-UNDO. The default is FALSE (NO-UNDO). The attribute can be set if there is no active transaction. Within an active transaction, it can be set as long as the temp-table is empty.

Data type: LOGICAL

Access: Readable/Writable

Applies to: Temp-table object handle

UNDO-THROW-SCOPE attribute

Indicates if UNDO, THROW has been set as the default error handling directive at either the routine level or the block level.

Data type: CHARACTER

Access: Read-only

Applies to: RCODE-INFO handle

The ROUTINE-LEVEL ON ERROR UNDO, THROW statement and the BLOCK-LEVEL ON ERROR UNDO, THROW statement override default error handling. The -undothrow startup parameter can also be used to override default error handling. See the *Startup Command and Parameter Reference* for more information about -undothrow.

This attribute returns "ROUTINE-LEVEL", or "BLOCK-LEVEL" to indicate which blocks are affected by UNDO, THROW error handling. It returns the empty string ("") when default error handling is in effect.

UNIQUE-ID attribute

A value, with one noted exception, that is guaranteed unique for a given handle object type within an ABL session.

Data type: INT64

Access: Read-only

Applies to: Buffer object handle, Buffer-field object handle, Procedure object handle, ProDataSet object handle, Query object handle, SAX-attributes object handle, SAX-reader object handle, SOAP-header object handle, SOAP-header-entryref object handle, SOURCE-PROCEDURE system handle, TARGET-PROCEDURE system handle, Temp-table object handle, THIS-PROCEDURE system handle, X-document object handle, X-noderef object handle

So, for example, the value of this attribute for a temp-table handle and a ProDataSet handle can be the same. However, the handles for no two temp-tables share this value.

The exception is the X-noderef object handle. The UNIQUE-ID attribute for an x-noderef handle is only unique within a given XML document.

Note: The value of the HANDLE attribute is guaranteed to be unique among the HANDLE attributes for all object instances in an ABL session.

UNIQUE-MATCH attribute
(Windows only; Graphical interfaces only)

Specifies that the combo-box widget automatically complete keyboard input based on a unique match to items in the drop-down list.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (column), COMBO-BOX widget

When the UNIQUE-MATCH attribute is TRUE, the widget's edit control compares the input to the items in the drop-down list. After each incremental character keystroke, the edit control searches through the items in the drop-down list for a unique match. When a unique match is found, the full item is displayed in the edit control. The automatically completed portion of the item is highlighted. You can replace the highlighted portion of the item by typing over it, or you can delete the highlighted portion of the item using the **DELETE** key or the **BACKSPACE** key. The default value is FALSE.

For browses, this attribute applies only to combo-box browse columns.

UPDATE-ATTRIBUTE( ) method

Finds an attribute and updates its value within a SAX-attributes object.

Return type: LOGICAL

Applies to: SAX-attributes object handle

Syntax

```
UPDATE-ATTRIBUTE ( { attribute-name | index } , attribute-value
  [ , namespaceURI ] ) )
```

attribute-name

A CHARACTER expression evaluating to the fully qualified or unqualified name of the attribute.

index

An integer expression evaluating to the position of the attribute in the attribute list. The first attribute has an index of 1. When using *index*, *namespaceURI* must be the Unknown value (?), an empty string, or not supplied. Otherwise, an invalid argument error will be raised.

attribute-value

A CHARACTER expression evaluating to the value of the attribute.

namespaceURI

A CHARACTER expression evaluating to:

- The URI of the attribute
- A zero-length string
- The Unknown value (?) if the attribute doesn't contain a namespace

Call this method to update an attribute value within a SAX-attributes object. For example, if an XML element obtained from the SAX-reader object need to be modified before being used to write the XML element to a document being written by the SAX-writer object, use this method to find and update the attributes.

The index values of attributes left in the list is then updated to reflect the removed attribute. The NUM-ITEMS value will also be decremented to reflect the removed attribute.

If the method fails to find a match, because the name is not found or the index position is invalid, then the method will return FALSE.

If using an *attribute-name*, the method updates the first match it finds in the list.

The following are examples:

```
DEFINE VARIABLE hSAX-attributes as HANDLE.

CREATE SAX-ATTRIBUTES hSAX-attributes.

hSAX-attributes:INSERT-ATTRIBUTE( "language", "EN" ). /* index = 1 */
hSAX-attributes:INSERT-ATTRIBUTE( "year", "2005" ).   /* index = 2 */

hSAX-attributes:UPDATE-ATTRIBUTE( "year", "2006").
hSAX-attributes:UPDATE-ATTRIBUTE( 2, "2007").
```

Note: The UPDATE-ATTRIBUTE( ) method allows some non-printing characters to be written to XML documents. These characters (for example, #x1, #x1F) are not part of the XML Version 1.0 standard, but are allowed in order to support ABL applications.

See also

INSERT-ATTRIBUTE( ) method, REMOVE-ATTRIBUTE( ) method

URL attribute

A URL to connect to an application server, through the APSV transport.

Data type: CHARACTER

Access: Read-only

Applies to: CODEBASE-LOCATOR system handle

Valid URL protocols depend on the LOCATOR-TYPE. If LOCATOR-TYPE is "AppServer", valid URL protocols include: HTTP, HTTPS, and AppServer. If LOCATOR-TYPE is "InternetServer", valid URL protocols include: HTTP, HTTPS, and FILE.

URL-DECODE( ) method

Returns a URL string to decode. This method is called by the `url-decode` WebSpeed API function. Intended for internal use only.

Return type: CHARACTER

Applies to: WEB-CONTEXT system handle

URL-ENCODE( ) method

Returns characters to encode. This method is called by the `url-encode` WebSpeed API function. Intended for internal use only.

Return type: CHARACTER

Applies to: WEB-CONTEXT system handle

URL-PASSWORD attribute

Password parameter for connecting to the server referenced in the URL, if required by the URL protocol.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: CODEBASE-LOCATOR system handle

URL-USERID attribute

Userid parameter for connecting to the server referenced in the URL, if required by the URL protocol.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: CODEBASE-LOCATOR system handle

USER-DATA ( ) method

Allows an application to write custom information to a special section at the end of the profiler output file. The data to be written is specified by the parameter value passed to the method call. If successful, USER-DATA() returns TRUE. If it fails to write the data, then it returns FALSE. A failure can occur only if the profiler is unable to open a temporary file. If the value passed is unknown, then a question mark (?) is written; otherwise, the value passed is enclosed in quotation marks. In addition to the string value, the profiler writes an integer value that represents the number of microseconds that elapsed since the start of the profiling process.

Return-type: LOGICAL

Applies to: PROFILER system handle

Syntax

```
USER-DATA ( string-expression  )
```

string-expression

A string expression to write custom information to a special section at the end of the profiler output file.

See also

WRITE-DATA( ) method

USER-ID attribute

The non-qualified user ID (user name) associated with a client-principal object. You must set this attribute before you can seal the associated client-principal object with the:

- SEAL( ) method as part of an application-performed user authentication operation
- SET-CLIENT( ) method or SET-DB-CLIENT function in an OpenEdge-performed user authentication operation

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

Any value you assign to this attribute cannot contain the domain delimiter character ("@"). You can assign the empty string ("") to this attribute. Any other requirements for setting this value depend on the authentication system used to authenticate the user identity. For more information on authentication systems and user ID's, see *Learn about Identity Management*.

Note: To authenticate the user identity for a valid setting of this attribute, the code page of the authenticating user account system must match the setting of the CPINTERNAL attribute on the SESSION system handle.

This attribute can be set using the INITIALIZE( ) method or the QUALIFIED-USER-ID attribute. Writing this attribute directly overwrites any value previously set by any mechanism.

If the attribute has never been initialized, reading it returns the empty string (""). Otherwise, it returns the most recent value set through the INITIALIZE( ) method, QUALIFIED-USER-ID attribute, or the USER-ID attribute, itself.

Once the client-principal object is sealed, this attribute is read-only, and attempting to write to it raises a run-time error.

ABL also raises ERROR if you attempt to assign:

- The Unknown value (?)
- Any character expression that contains the domain delimiter character ("@")
- A character expression when the client-principal object is sealed and in the LOGIN, LOGOUT, EXPIRED, or FAILED state (see the LOGIN-STATE attribute entry)

See also

DOMAIN-NAME attribute, INITIALIZE( ) method, QUALIFIED-USER-ID attribute

V6DISPLAY attribute
(Windows only)

Indicates whether the AVM follows Progress Version 6 rules or Progress Version 7 rules when it lays out and displays widgets in Windows. This attribute lets you compile and execute Progress Version 6 applications on Progress Version 7 in Windows.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

If the V6DISPLAY attribute is TRUE, the AVM uses Version 6 rules to manage the display. That is:

- The default font is the default system fixed pitch font (overridable using the DefaultFixedFont parameter in the current environment, which might be the Registry (Windows only) or an initialization file).
- All fill-ins have no borders.
- Fill-ins enabled for input use an underline version of the system fixed pitch font (overridable using the DefaultUpdateFont parameter in the current environment, which might be the Registry (Windows only) or an initialization file).
- The default window size (row/column) is 25 by 80 (overridable in the current environment, which might be the Registry (Windows only) or an initialization file).

Note: PUT SCREEN output is not restorable in graphical environments.

To run an application with V6DISPLAY set to TRUE, you must compile the application with the V6DISPLAY set to TRUE.

Note:

- The OpenEdge ADE toolset was not compiled or designed to run in V6DISPLAY mode. Running the OpenEdge ADE in V6DISPLAY mode may result in clipped display elements and other unexpected behavior.
- Setting V6DISPLAY to TRUE when running the OpenEdge ADE toolset may also degrade application compilation performance.

This attribute provides the same functionality as the V6Display parameter in the current environment, which might be the Registry (Windows only) or an initialization file. For more information on environments, see the topics on user interface environments in *Manage ABL Applications*.

VALIDATE( ) method

Executes any validation tests established in a database or specified by the VALIDATE option of the Format phrase.

Return type: LOGICAL

Applies to: BROWSE widget (browse and column), COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

Syntax

```
VALIDATE ( [ "ENABLED-FIELDS" ] )
```

`"ENABLED-FIELDS"`

Validate enabled fields only.

If this option does not appear, the VALIDATE method validates all fields, whether enabled or not.

For a supported field-level widget, this method executes the validation test associated with the underlying field or variable.

For a frame or dialog box, this method executes the validation tests for every supported field-level widget in the frame or dialog box (except the browse, which you must VALIDATE explicitly). If the test for any field-level widget in the frame fails, the AVM displays the validation message and gives focus to the first widget in the frame or dialog box that is both visible and sensitive and whose data has failed validation.

For a browse, VALIDATE executes all validation tests associated with the browse and its children.

Note: During data entry, any widget that receives input focus is always validated. This method allows your procedure to validate any and all widgets in a frame, whether or not they currently have input focus.

If the validation is successful, the method returns TRUE. Otherwise, it returns FALSE.

VALIDATE-DOMAIN-ACCESS-CODE( ) method

Returns TRUE if the encrypted value of the domain access code matches the clear text value.

Return-type: LOGICAL

Applies to: SECURITY-POLICY system handle

Syntax

```
VALIDATE-DOMAIN-ACCESS-CODE ( encrypted-value , text-value )
```

encrypted-value

A CHARACTER expression containing the encrypted value of the domain access code.

text-value

A CHARACTER expression containing the clear text value of the domain access code.

See also

ENCODE-DOMAIN-ACCESS-CODE( ) method

VALIDATE-EXPRESSION attribute

The value of the validation expression in the database schema for the database field that corresponds to the buffer-field.

The VALIDATE-EXPRESSION attribute lets you write user input validation code for interfaces that the AVM's automatic user input validation does not support.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer-field object handle

If a buffer-field is associated with a dynamic browse column, you should set the buffer-field's VALIDATE-EXPRESSION attribute before the dynamic browse column is added to the browser (via ADD-LIKE-COLUMN( )). The validation expression is compiled at this time. If the VALIDATE-EXPRESSION attribute is changed later, it is ignored.

VALIDATE-MESSAGE attribute

The value of the validation message in the database schema for the database field that corresponds to the buffer-field.

The VALIDATE-MESSAGE attribute lets you write user input validation code for interfaces that the AVM's automatic user input validation does not support.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer-field object handle

VALIDATE-SEAL( ) method

Validates the message authentication code (MAC) generated by the SEAL( ) method to seal a client-principal object.

You can use this method to validate the seal whenever necessary.

Return type: LOGICAL

Applies to: Client-principal object handle

Syntax

```
VALIDATE-SEAL ( [ domain-access-code ] )
```

domain-access-code

An optional character expression containing the access code that is defined for the user's domain and that originally was used to seal the client-principal object. The AVM converts this value to a UTF-8 before using it, which ensures a consistent value regardless of code page settings.

ABL raises ERROR if the client-principal object is not:

- Sealed
- In the LOGIN state

If you specify a domain access code, the AVM uses the specified value to validate the seal.

Caution: Given any client-principal that is sealed with it, this domain access code provides session and database access when validated using a single sign-on (SSO) operation against a trusted domain registry that contains a matching domain with the same access code. Therefore, Progress Software corporation **strongly** recommends that you take steps to hide or otherwise protect any domain access code that you specify in your ABL code from access (hacking) by unauthorized users.

If you do not specify a domain access code, the AVM uses the access code defined for the domain in the trusted domain registry to validate the seal.

The AVM validates the seal by comparing it to the MAC generated from either the specified domain access code or the matching domain access code stored in the trusted domain registry. If the seal matches the generated MAC, the seal is valid and this method returns TRUE. Otherwise, the seal is invalid and this method returns FALSE.

The AVM also checks the LOGIN-EXPIRATION-TIMESTAMP attribute. If the client-principal object expires before you can validate its seal, the AVM sets the LOGIN-STATE attribute to `"EXPIRED"` and returns FALSE.

VALIDATE-SEAL( ) only generates an audit event when the method finds the client principal to be expired for the first time. When this happens, the method generates an audit event for a logout operation.

The following code fragment illustrates how to use the VALIDATE-SEAL( ) method:

```
DEFINE VARIABLE hCP    AS HANDLE    NO-UNDO.
DEFINE VARIABLE key    AS CHARACTER NO-UNDO.
DEFINE VARIABLE val-ok AS LOGICAL   NO-UNDO.
. . .
CREATE CLIENT-PRINCIPAL hCp.
. . .
val-ok = hCP:VALIDATE-SEAL(key).
```

See also

LOGIN-EXPIRATION-TIMESTAMP attribute, LOGIN-STATE attribute, SEAL( ) method

VALIDATE-XML attribute

Sets validation on parsing when an XML document is posted to the transaction server. The default is NO.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

VALIDATION-ENABLED attribute

Indicates whether the parser validates the XML document against the DTD.

Data type: LOGICAL

Access: Readable/Writable

Applies to: SAX-reader object handle

TRUE indicates that the parser validates the XML document against the DTD. The default is TRUE. Validation errors are caught in a callback event procedure named **`error`**. If this callback procedure is not implemented, then validation errors cannot be caught.

Note: If VALIDATION-ENABLED is FALSE, the parser still checks that the XML document is well formed.

VALUE attribute

The data values in the system clipboard.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: CLIPBOARD system handle

During single-item operation (with the CLIPBOARD:MULTIPLE attribute set to FALSE), reading this attribute returns all the data in the clipboard, and writing to attribute completely replaces any and all data in the clipboard. During multiple-item operation (with the CLIPBOARD:MULTIPLE attribute set to TRUE), reading this attribute returns one of several data items in the clipboard, and writing to this attribute appends a data item to a buffered value formatted to replace the data in the clipboard.

Note: In Windows, the clipboard can store a maximum of 64K of data.

If there is no data in the clipboard or the last data item has been read during a multiple-item operation, the VALUE attribute returns the Unknown value (?). Reading this attribute during either single-item or multiple-item operation has no effect on the existing clipboard data value(s). Setting the VALUE attribute to the Unknown value (?) during single- or multiple-item operation has no effect. To clear the clipboard of all data, set the VALUE attribute to the null string in a single-item operation. For more information on using the VALUE attribute, see the reference entry for the CLIPBOARD system handle.

VERSION attribute

Determines the value of the version string in the XML declaration of a SAX-writer object.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SAX-writer object handle

You can set the attribute to the value of the version string in the XML declaration. The default value is `"1.0"`. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
```

You can read this attribute at all times, but you can only write to it when the WRITE-STATUS is either SAX-WRITE-IDLE or SAX-WRITE-COMPLETE. That is, it can only be changed when the writer is not writing, otherwise it fails and generates an error message.

If STRICT is TRUE, the only valid version is `"1.0"`, and the VERSION attribute cannot be changed or you get an error message. If STRICT is FALSE, the version in the prolog of the XML document will match the value of the VERSION attribute.

If the value is an empty string (`""`), then the version string will not appear in the XML declaration.

VIEW-AS attribute

Sets the widget type of a dynamic or static browse column for a buffer-field object.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer-field object handle

Set this attribute to specify the widget type for a dynamic browse column (that is, a column created with the ADD-LIKE-COLUMN( ) method or ADD-COLUMNS-FROM( ) method). Valid values are `"COMBO-BOX"`, `"DROP-DOWN"`, `"DROP-DOWN-LIST"`, `"FILL-IN"` and `"TOGGLE-BOX"`. The default value is `"FILL-IN"`.

Note: In a character interface, the widget type displayed for browse columns is always FILL-IN no matter what value you specify for this attribute.

As an added convenience, you can also create a combo-box browse column by setting the buffer-field's VIEW-AS attribute to `"COMBO-BOX"` and setting its SUBTYPE attribute to either `"DROP-DOWN"` or `"DROP-DOWN-LIST"`.

A widget type of `"COMBO-BOX"` or `"DROP-DOWN-LIST"` is valid only when the buffer-field's data type is CHARACTER, DATE, DECIMAL, INTEGER, INT64 or LOGICAL. A widget type of `"DROP-DOWN"` is valid only when the buffer-field's data type is CHARACTER. A widget type of `"TOGGLE-BOX"` is valid only when the buffer-field's data type is LOGICAL. If the widget type is not valid for the buffer-field's data type, the AVM generates a run-time error.

Setting this attribute to the Unknown value (?) restores the default value of `"FILL-IN"`.

You can query this attribute for the widget type of a dynamic or static browse column.

For any type of combo-box browse column, this attribute returns `"COMBO-BOX"`. Refer to the column's SUBTYPE attribute to determine whether the combo-box is a DROP-DOWN or DROP-DOWN-LIST combo-box.

VIEW-FIRST-COLUMN-ON-REOPEN attribute

Controls whether the browse, when an OPEN Query statement is run, displays the first column in the viewport or the columns that were in the viewport before the Query was reopened.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

When the VIEW-FIRST-COLUMN-ON-REOPEN attribute is set to:

- TRUE, and the query for a browse is reopened, the browse displays the first row of data and the first column in the leftmost position
- FALSE, and the query for a browse is reopened, the browse displays the first row of data and the columns that were displayed in the viewport before the query was reopened

  For example, if this attribute is set to FALSE and a user had scrolled to the far-right column, the next time the browse for a query is reopened, the browse displays the first row of data and the far-right column.

The default value is FALSE.

VIRTUAL-HEIGHT-CHARS attribute

The maximum height of the widget, in character units.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget, WINDOW widget

For a non-scrollable frame, VIRTUAL-HEIGHT-CHARS has the same value as the HEIGHT-CHARS attribute. For a scrollable frame, VIRTUAL-HEIGHT-CHARS specifies the height of the entire frame while HEIGHT-CHARS specifies the height of the visible portion of the frame.

VIRTUAL-HEIGHT-PIXELS attribute

The maximum height of the widget, in pixels.

Data type: INTEGER

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget, WINDOW widget

For a non-scrollable frame, VIRTUAL-HEIGHT-PIXELS has the same value as the HEIGHT-PIXELS attribute. For a scrollable frame, VIRTUAL-HEIGHT-PIXELS specifies the height of the entire frame while HEIGHT-PIXELS specifies the height of the visible portion of the frame.

VIRTUAL-WIDTH-CHARS attribute

The maximum width of the widget, in character units.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget, WINDOW widget

For a non-scrollable frame, VIRTUAL-WIDTH-CHARS has the same value as the WIDTH-CHARS attribute. For a scrollable frame, VIRTUAL-WIDTH-CHARS specifies the width of the entire frame while WIDTH-CHARS specifies the width of the visible portion of the frame.

VIRTUAL-WIDTH-PIXELS attribute

The maximum width of the widget, in pixels.

Data type: INTEGER

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget, WINDOW widget

For a non-scrollable frame, VIRTUAL-WIDTH-PIXELS has the same value as the WIDTH-PIXELS attribute. For a scrollable frame, VIRTUAL-WIDTH-PIXELS specifies the width of the entire frame while WIDTH-PIXELS specifies the width of the visible portion of the frame.

VISIBLE attribute

Indicates whether a widget is currently visible on the display.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DEBUGGER system handle, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

This attribute is read-only for field groups, menus, menu items, and submenus. A field-level widget must be specified in a frame definition before you set its VISIBLE attribute.

The behavior of the VISIBLE attribute depends on the setting of the HIDDEN attribute of related widgets:

- When you set the VISIBLE attribute of a window to TRUE:
  - The AVM displays that window and the widgets it contains whose VISIBLE attributes are already set to TRUE. Otherwise, you must explicitly DISPLAY or VIEW a widget, or otherwise set a widget's VISIBLE attribute to TRUE in order to display it in the window.
  - The AVM displays that window and all ancestor windows only if no ancestor window has its HIDDEN attribute set to TRUE. If the AVM displays the window, it also displays all descendant windows down to, but not including, the first descendant window whose HIDDEN attribute is set to TRUE.
- When you set the VISIBLE attribute of any widget within a window to TRUE, the AVM displays that widget, any ancestor frames, and the window (if necessary), unless the HIDDEN attribute of the window is TRUE. If the window's HIDDEN attribute is TRUE, the AVM sets the VISIBLE attributes of the widget and any ancestor frames to TRUE and sets the HIDDEN attributes of the widget and its ancestor frames to FALSE without displaying them.
- When you set the VISIBLE attribute of a frame to TRUE, the AVM displays all of its field-level widgets and descendant frames, except those whose HIDDEN attributes are TRUE.
- When you explicitly set the VISIBLE attribute of any widget to TRUE, the AVM sets its HIDDEN attribute to FALSE. If you explicitly set the VISIBLE attribute of a field-level widget or child frame to FALSE while its parent frame remains visible, the AVM also sets the HIDDEN attribute of the field-level widget or child frame to TRUE. If you explicitly set the VISIBLE attribute of a child window to FALSE, the HIDDEN attribute of the child window remains unchanged, whether or not the parent window is visible.
- The following behavior is true for the browse column:
  - The syntax of the VISIBLE attribute for the browse column is as follows:

```
VISIBLE [ IN BROWSE browse-name]
```

  - The behavior of the VISIBLE attribute for a browse column does not depend on the setting of the HIDDEN attribute of the related widget.
  - Changing the VISIBLE attribute of a browse column may affect which columns are locked if NUM-LOCKED-COLUMNS has been set. This is because NUM-LOCKED-COLUMNS only applies to visible columns. For example, if the first three columns of a browse are locked and the second column is made not VISIBLE, the fourth column will then become locked.
  - If a widget is not already realized and you set its VISIBLE attribute to TRUE, the AVM realizes that widget.
  - In character mode, the VISIBLE attribute is always set to TRUE.

WARNING attribute

Indicates whether the last compilation produced warning messages.

Data type: LOGICAL

Access: Read-only

Applies to: COMPILER system handle

If the WARNING attribute is TRUE, there were warning messages from the last compilation.

WC-ADMIN-APP attribute

Indicates whether a WebClient application is an Administrator installation or a personal installation. If TRUE, a user with Administrator privileges installed the application. If FALSE, the user did not have those privileges and installed a personal instance of the application.

Data type: LOGICAL

Access: Read-only

Applies to: SESSION system handle

This attribute is only valid for WebClient sessions. The LIST-QUERY-ATTRS function's output only shows it for WebClient sessions.

WHERE-STRING attribute

Returns and allows you to modify the current WHERE expression from the query generated for a specified data-relation that links the child table to its parent.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Data-relation object handle

This attribute evaluates to the WHERE expression from the query that the AVM generates for you based on the data-relation between parent and child buffers. You can also use this attribute to build an extended query of your own based on this default relationship.

Note: You can provide the initial WHERE expression for a query using the FILL-WHERE-STRING attribute.

If you assign a new value to the attribute, the AVM uses the new WHERE expression when it re-opens the query during navigation to filter the child table of the data-relation. When you set this attribute, if the AVM finds an error in the new WHERE expression, it raises a run-time error on the assignment.

The NAME attribute of the Temp-table object handle is writeable for dynamic and AVM-generated temp-tables. You might need to update a WHERE-STRING that references a renamed temp-table with new strings using the new table name.

Typically, you modify an existing WHERE expression by appending additional conditions to it, as shown in the `r-wherestr.p` example procedure. This example provides the initial WHERE expression using the FILL-WHERE-STRING attribute. It then generates the complete initial query for the data-relation with the call to the ProDataSet FILL( ) method, and uses the WHERE-STRING attribute to modify the existing WHERE expression to further filter the query buffers.

r-wherestr.p

```
DEFINE TEMP-TABLE ttCus LIKE Customer.
DEFINE TEMP-TABLE ttOrder LIKE Order.
DEFINE DATASET dsCus FOR ttCus,ttOrder DATA-RELATION dr1
  FOR ttCus, ttOrder RELATION-FIELDS(CustNum,CustNum).

DEFINE DATA-SOURCE dscCus FOR Customer.
DEFINE DATA-SOURCE dscOrd FOR Order.

DEFINE QUERY q1 FOR ttCus.
DEFINE QUERY q2 FOR ttOrder.

DEFINE BROWSE b1 QUERY q1
  DISPLAY ttCus.CustNum ttCus.Name ttCus.Address WITH 3 DOWN.
DEFINE BROWSE b2 QUERY q2
  DISPLAY ttOrder.CustNum ttOrder.OrderNum ttOrder.ShipDate WITH 6 DOWN.

BUFFER ttCus:ATTACH-DATA-SOURCE(DATA-SOURCE dscCus:HANDLE).
BUFFER ttOrder:ATTACH-DATA-SOURCE(DATA-SOURCE dscOrd:HANDLE).
DATA-SOURCE dscCus:FILL-WHERE-STRING = "WHERE CustNum < 3".
DATASET dsCus:FILL( ).

DATASET dsCus:GET-RELATION(1):WHERE-STRING
  = DATASET dsCus:GET-RELATION(1):WHERE-STRING
    + " AND ttOrder.OrderNum < 100".

/* Shows full dataset */
FOR EACH ttCus:
  DISPLAY ttCus.CustNum.
  FOR EACH ttOrder OF ttCus:
    DISPLAY ttOrder.OrderNum ttOrder.OrderDate.
  END.
END.

BROWSE b1:QUERY = DATASET dsCus:TOP-NAV-QUERY.
BROWSE b2:QUERY = DATASET dsCus:GET-RELATION(1):QUERY.

/* Shows only filtered records */
ENABLE b1 b2 WITH FRAME frX ROW 1 SIZE 70 BY 30.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

See also

FILL-WHERE-STRING attribute, QUERY attribute

Widget-Handle property
(Windows only; Graphical interfaces only)

The handle of the control frame associated with the control-frame COM object.

Return type: HANDLE

Access: Read-only

Applies to: CONTROL-FRAME widget, COM object

Note: References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the Accessing COM object properties and methods.

WIDGET-ENTER attribute

A handle, in a trigger associated with an ENTRY event or a LEAVE event, to the next widget to receive input focus.

Data type: HANDLE

Access: Read-only

Applies to: LAST-EVENT system handle

The WIDGET-ENTER attribute is meaningful only within an ENTRY or LEAVE trigger.

For browse widgets, WIDGET-ENTER is different depending on whether the browse is editable or read-only. For editable browse widgets, WIDGET-ENTER contains the handle of the column with focus. For read-only browse widgets, WIDGET-ENTER contains the handle of the browse.

WIDGET-ID attribute
(Windows only; Graphical interfaces only)

An application-defined widget ID for a static or dynamic widget. The value of this attribute must be an even INTEGER value between 2 and 65534, inclusive, and it must be unique across all widget IDs in a window.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

Specify an application-defined widget ID when you want to identify the widget at run time while testing your ABL GUI application with a third-party automated test tool. When specified, the AVM uses this application-defined widget ID when creating the widget at run time, instead of using the widget ID it normally generates by default. The widget ID value of any given widget remains the same across ABL sessions, unless you purposely change the value of the widget ID using this attribute. This allows a third-party automated test tool to identify the same widget consistently each time you run the tool with your application.

To enable application-defined widget IDs in your ABL GUI application, you must specify the Use Widget ID (`-usewidgetid`) startup parameter. For more information about this startup parameter, see *Startup Command and Parameter Reference*.

If you do not specify the Use Widget ID (`-usewidgetid`) startup parameter, and your application contains application-defined widget IDs, the AVM ignores any WIDGET-ID attribute or option settings and generates its own widget IDs. In this case, querying the WIDGET-ID attribute returns the Unknown value (?).

If you specify the Use Widget ID (`-usewidgetid`) startup parameter and your application contains application-defined widget IDs, or a combination of application-defined and ABL-defined widget IDs, the AVM honors any application-defined widget IDs and assigns ABL-defined widget IDs as necessary. In this case, querying the WIDGET-ID attribute returns the assigned widget ID (whether it be an application-defined value or an ABL-defined value).

You can set this attribute only before the widget is realized. Once the widget is realized, this attribute is read-only.

For frames, the AVM uses this attribute value as the basis for assigning a unique widget ID for each child widget within the frame by combining the frame widget ID with the child widget ID. For example, a frame defined with a widget ID of 100 that contains a fill-in widget defined with a widget ID of 2 results in a fill-in widget with a widget ID of 102 at run time. In this way, the widget ID of each child widget within a given frame is unique within that frame, as well as across multiple instances of that frame within a given window. If the value of the frame widget ID combined with the child widget ID is greater than 65534, the AVM displays a warning message and assigns a unique widget ID to the child widget.

When a frame is displayed as a down frame, you can specify an application-defined widget ID for the first instance of the widget on the down frame. The AVM assigns a unique widget ID for each additional instance of the widget on the down frame (based on the number of iterations in the down frame) using consecutive even numbers. Likewise, when a frame contains extent fields, the AVM assigns a widget ID to each of the extent elements in the frame.

When a frame is displayed as a dialog box, which is a special type of frame displayed in its own window, the widget ID for any child widgets inside the dialog box must be unique only within that dialog box.

Note: The default widget ID for a frame is 0 (zero). Use caution when defining multiple frames, or multiple instances of a frame, in a single window and allowing the widget IDs to default to 0, because you are more likely to encounter a duplicate widget ID conflict.

For radio-set widgets, which are built with individual radio buttons, the AVM uses this attribute value as the basis for assigning a unique widget ID for each radio button of the given radio-set widget using consecutive even numbers.

For browse widgets, the AVM uses this attribute value as the basis for assigning a unique widget ID for each column within the browse by automatically incrementing the browse widget ID by 1 for each column within the browse sequentially from left to right.

ABL also provides for assigning widget IDs to widget labels by reserving the previous odd value of each widget ID for the widget's label. For example, if you assign a widget a widget ID of 10, ABL reserves widget ID 9 for the widget's label. ABL does not provide for assigning widget IDs to browse column labels.

You cannot specify an application-defined widget ID for FIELD-GROUP, LITERAL, MENU, MENU-ITEM, SUB-MENU, or WINDOW widgets.

If you specify an invalid widget ID value in a static widget definition, ABL generates a compiler error whether the Use Widget ID (`-usewidgetid`) startup parameter is specified or not. If you specify an invalid widget ID value in a dynamic widget definition, the AVM generates a run-time error only when the startup parameter is specified.

Caution: To avoid duplicate widget ID conflicts, within and across multiple instances of a widget in a single window, be sure to specify widget IDs within numeric ranges that take other widgets into account. For example, do not specify frame widget IDs in multiples of 10 when you have one or more frame widgets that contain more than 9 child widgets because it will result in a duplicate ID conflict.

WIDGET-LEAVE attribute

A handle, in a trigger associated with an ENTRY event or a LEAVE event, to the widget that had input focus prior to the event.

Data type: HANDLE

Access: Read-only

Applies to: LAST-EVENT system handle

The WIDGET-ENTER attribute is meaningful only within an ENTRY or LEAVE trigger.

For browse widgets, WIDGET-LEAVE is different depending on whether the browse is editable or read-only. For editable browse widgets, WIDGET-LEAVE contains the handle of the column just left. For read-only browse widgets, WIDGET-LEAVE contains the handle of the field-level widget just left.

Width property
(Windows only; Graphical interfaces only)

The width of the control-frame and control-frame COM object, in pixels.

Return type: INTEGER

Access: Readable/Writeable

Applies to: CONTROL-FRAME widget, COM object

Setting this value changes the WIDTH-CHARS attribute and WIDTH-PIXELS attribute of the corresponding control-frame widget to an equivalent value.

Note: References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the Accessing COM object properties and methods.

WIDTH-CHARS attribute

The width of the widget or the display used in the current session, in character units.

Data type: DECIMAL

Access: Readable/Writable

Applies to: BROWSE widget (browse, column, and cell), Buffer-field object handle, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SESSION system handle, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

The attribute is read-only for field groups, and the SESSION handle.

For control-frames, the WIDTH-CHARS attribute maps to the Width property of the control-frame COM object (ActiveX control container).

In character mode, the WIDTH-CHARS attribute must be set before the browse is realized. Attempting to set the attribute after realization of the browse causes a run-time error.

For editor widgets, this attribute can set the word wrap margin for the WORD-WRAP attribute. For more information, see the WORD-WRAP attribute reference entry.

For buffer-field objects, the WIDTH-CHARS attribute is the number of characters in the STRING-VALUE, which the AVM calculates using the FORMAT attribute. In addition, the WIDTH-CHARS attribute of a buffer-field is readable but not writeable.

For browses, the WIDTH-CHARS attribute sets the width, in characters, of the browse without changing the width of any browse column. If you change the value of a browse's WIDTH-CHARS or WIDTH-PIXELS attribute, the horizontal scrollbar might appear or disappear, which might cause the number of rows that appear in the viewport to change.

For browse columns, the WIDTH-CHARS attribute sets the width, in characters, of the browse column without changing the width of the browse.

WIDTH-PIXELS attribute

The width of the widget or the screen display used in the current session, in pixels.

Data type: INTEGER

Access: Readable/Writable

Applies to: BROWSE widget (browse, column, and cell), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SESSION system handle, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

The attribute is read-only for field groups, and the SESSION handle.

In character mode, the WIDTH-PIXELS attribute must be set before the browse is realized. Attempting to set the attribute after realization of the browse causes a run-time error.

For control-frames, the WIDTH-PIXELS attribute maps to the Width property of the control-frame COM object (ActiveX control container).

For editor widgets, this attribute can set the word wrap margin for the WORD-WRAP attribute. For more information, see the WORD-WRAP attribute reference entry.

For browses, the WIDTH-PIXELS attribute sets the width, in pixels, of the browse without changing the width of any browse column. If you change the value of a browse's WIDTH-CHARS or WIDTH-PIXELS attribute, the horizontal scrollbar might appear or disappear, which might cause the number of rows that appear in the viewport to change.

For browse columns, the WIDTH-PIXELS attribute sets the width, in pixels, of the browse column without changing the width of the browse.

WINDOW attribute

A handle to the window that owns a widget or that contains the owner of a widget.

Data type: HANDLE

Access: Read-only

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

For a menu bar or pop-up menu of a window, the WINDOW and OWNER attributes have the same value. For a window, the WINDOW attribute returns the window's handle (not its parent's handle, if any). For a menu bar or pop-up menu of a window, the WINDOW and OWNER attributes have the same value.

WINDOW-STATE attribute

The current visual state of a window in the window system.

Data type: INTEGER

Access: Readable/Writeable

Applies to: WINDOW widget

The possible values can be expressed as compiler constants. The following table lists these values.

Table 110: Window state values

Compiler constant	Value	Description
WINDOW-MAXIMIZED	1	The window is maximized to fill the entire display.
WINDOW-MINIMIZED	2	The window is minimized (iconified).
WINDOW-NORMAL	3	The window is in the "restored" state. Initially, this refers to a state that is neither maximized nor minimized. However, setting WINDOW-STATE to WINDOW-NORMAL restores the window to its previous state, which may be the maximized, minimized, or neither.
WINDOW-DELAYED-MINIMIZE	4	The window is minimized (iconified) the next time a new window, dialog box, or alert box is displayed. This differs from setting WINDOW-STATE to WINDOW-MINIMIZED, which minimizes the window immediately.

You can change the state of a window programmatically by setting the WINDOW-STATE attribute. Note that you can change a window to its maximized state in Windows only.

WINDOW-SYSTEM attribute

A value that indicates the windowing system the application is using.

Data type: CHARACTER

Access: Read-only

Applies to: SESSION system handle

- For graphical interfaces in Windows:
  - If the display is set to the Windows XP Theme, and a manifest file is used, the value is `"MS-WINXP"`.
  - If the display is set to the Windows XP Theme, and a manifest file is not used, the value is `"MS-WIN95"`.
  - If the display is set to the Windows Classic Theme, the value is `"MS-WIN95"`.
- For character interfaces, the value is `"TTY"`.

ABL supports an override option that enables applications that need the WINDOW-SYSTEM attribute to return the value of "MS-WINDOWS" for all Microsoft operating systems to do so. To establish this override capability, define the WindowSystem key in the Startup section in the current environment, which might be the registry or an initialization file. If the WindowSystem key is located, the WINDOW-SYSTEM attribute returns the value associated with the WindowSystem key on all platforms.

WORD-WRAP attribute
(Graphical interfaces only)

Indicates whether word wrapping is enabled for an editor widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: EDITOR widget

If WORD-WRAP is TRUE, the editor automatically breaks lines at any word that crosses the word wrap margin of the text area. If WORD-WRAP is FALSE, the editor continues lines beyond the editor border up to the first hard return, and scrolls as required to keep the entered text in view. The user can scroll left and right to view the entire line. The default value for WORD-WRAP is TRUE.

In graphical interfaces, the word wrap margin is set by the WIDTH-CHARS, WIDTH-PIXELS, or INNER-CHARS attribute. In character interfaces, the word wrap margin is determined by either the WIDTH-CHARS or BUFFER-CHARS attribute, whichever is larger.

In Windows, both the regular editor and the large editor support WORD-WRAP.

Note: If the SCROLLBAR-HORIZONTAL attribute is set to TRUE, then WORD-WRAP is automatically set to FALSE. Likewise, if you set the WORD-WRAP attribute to TRUE, then SCROLLBAR-HORIZONTAL is automatically set to FALSE.

You can set this attribute only before the widget is realized.

WORK-AREA-HEIGHT-PIXELS attribute

Indicates the height of the work-area in pixels. The work-area is the portion of the Windows desktop that is not hidden by task bars. That is, the dimensions of the work-area are the dimensions of the Windows desktop minus the dimensions of all task bars on the Windows desktop.

Data type: INTEGER

Access: Read-only

Applies to: SESSION system handle

In character interfaces, this attribute returns the Unknown value (?).

WORK-AREA-WIDTH-PIXELS attribute

Indicates the width of the work-area in pixels. The work-area is the portion of the Windows desktop that is not hidden by task bars. That is, the dimensions of the work-area are the dimensions of the Windows desktop minus the dimensions of all task bars on the Windows desktop.

Data type: INTEGER

Access: Read-only

Applies to: SESSION system handle

On character platforms, this attribute returns the Unknown value (?).

WORK-AREA-X attribute

The starting x-coordinate (the upper left-hand corner) of the work-area in pixels. The work-area is the portion of the Windows desktop that is not hidden by task bars. That is, the dimensions of the work-area are the dimensions of the Windows desktop minus the dimensions of all task bars on the Windows desktop.

Data type: INTEGER

Access: Read-only

Applies to: SESSION system handle

On character platforms, this attribute returns the Unknown value (?).

WORK-AREA-Y attribute

The starting y-coordinate (the upper left-hand corner) of the work-area in pixels. The work-area is the portion of the Windows desktop that is not hidden by task bars. That is, the dimensions of the work-area are the dimensions of the Windows desktop minus the dimensions of all task bars on the Windows desktop.

Data type: INTEGER

Access: Read-only

Applies to: SESSION system handle

On character platforms, this attribute returns the Unknown value (?).

WRITE( ) method (Socket)

Writes data to the socket.

Return type: LOGICAL

Applies to: Socket object handle

Syntax

```
WRITE ( buffer , position , bytes-to-write )
```

buffer

A MEMPTR expression which contains data which should be written to the socket.

position

An integer expression greater than 0 that indicates the starting byte position within *buffer* which should be written to the socket.

bytes-to-write

An integer expression that specifies the number of bytes to be written to the socket.

WRITE( ) returns TRUE if the write operation succeeded normally and returns FALSE otherwise. An error can occur if:

- The position parameter is not greater than 0
- The amount of information requested to write exceeds the amount of data in the buffer
- Writing to the socket fails

This method expects *buffer* to identify a MEMPTR variable which already has a region of memory associated with it. The developer must call the SET-SIZE statement to allocate memory and associate it with a MEMPTR variable. It is the responsibility of the developer to free this memory, also via the SET-SIZE statement. The WRITE method will fail if the size of *buffer* is less than *bytes-to-write*.

Even if the WRITE( ) method returns TRUE, not all the bytes may have actually been written. To find out how many bytes were written, check the BYTES-WRITTEN attribute.

WRITE-CDATA( ) method

Adds a CDATA block to an XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-CDATA ( CDATA_block )
```

CDATA_block

A CHARACTER or LONGCHAR expression evaluating to the value of the CDATA block.

Call this method to add character data to the XML document. Character data in an XML document belongs exclusively in leaf nodes. (A leaf node is a bottom node; one that does not have any child nodes in a hierarchical tree structure, like an XML document.) Character data cannot appear outside of the root (document) node. The SAX-writer puts the block into the format of a CDATA section by adding the correct open and close tags using this syntax:

Syntax

```
<![CDATA[ CDATA_block ]]>
```

This method does not change the WRITE-STATUS attribute.

Note: The character data is serialized and not escaped by the SAX-writer.

Note: The WRITE-CDATA( ) method allows some non-printing characters to be written to XML documents. These characters (for example, #x1, #x1F) are not part of the XML Version 1.0 standard, but are allowed in order to support ABL applications.

WRITE-CHARACTERS( ) method

Adds character data to an XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-CHARACTERS ( { chardata | longchar } )
```

chardata

An expression that evaluates to a CHARACTER variable that contains the XML text.

longchar

An expression that evaluates to a LONGCHAR variable that contains the XML text.

Call this method to add character data to the XML document. Character data in an XML document belongs exclusively in leaf nodes. (A leaf node is a bottom node; one that does not have any child nodes in a hierarchical tree structure, like an XML document.) Character data cannot appear outside of the root (document) node.

This method sets the status to SAX-WRITE-CONTENT.

The method escapes all special characters according to the XML specification. For example, `"<"` is changed to `"<"`.

If the STRICT attribute is TRUE and the call would result in CHARACTER data being written at the document level (that is, outside of the root node), then the method fails.

Note: The WRITE-CHARACTERS( ) method allows some non-printing characters to be written to XML documents. These characters (for example, #x1, #x1F) are not part of the XML Version 1.0 standard, but are allowed in order to support ABL applications.

WRITE-COMMENT( ) method

Adds a comment to the XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-COMMENT ( value )
```

value

A CHARACTER or LONGCHAR expression evaluating to the text of the comment.

Call this method to add a comment node to the XML document. You can add comments at any time. The SAX-writer object creates the comment by enclosing the CHARACTER expression in open and close comment markers (<!-- and --!>). For example, with *value* set to "This is a comment", this method creates the following comment line:

```
<!-- This is a comment --!>
```

WRITE-DATA( ) method

Instructs the profiler to analyze the data that was recorded and to write the current information to the profile output file. If the write is successful, then the profiler returns TRUE. It returns FALSE if it fails to write the data. A failure to write data could mean either there is no new data or that the profiler could not open the specified output file. Time spent analyzing and writing the profiling data is deducted from any future data profiling.

Return-type: LOGICAL

Applies to: PROFILER system handle

Syntax

```
WRITE-DATA()
```

See also

COVERAGE attribute, DESCRIPTION attribute, FILE-NAME attribute, ENABLED attribute, LISTINGS attribute, PROFILING attribute, STATISTICS attribute, TRACING attribute and USER-DATA ( ) method

WRITE-DATA-ELEMENT( ) method

Creates a complete XML node in a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-DATA-ELEMENT ( name , { chardata | longchar } [ , namespace-URI ]
  [ , SAX-attributes-handle ] )
```

name

A CHARACTER or LONGCHAR expression evaluating to the fully qualified or unqualified name of the element.

chardata

An expression that evaluates to a CHARACTER variable that contains the XML text.

longchar

An expression that evaluates to a LONGCHAR variable that contains the XML text.

namespace-URI

A CHARACTER or LONGCHAR expression evaluating to the URI of the element, or an empty string ("") or the Unknown value (?).

SAX-attributes-handle

A HANDLE to a SAX-attributes object. The attributes in the SAX-attributes object are added to the new XML element.

Creates a complete XML node. This method call sets the WRITE-STATUS to SAX-WRITE-ELEMENT.

If you use *namespace-URI*, then the prefix is resolved in the following order:

1. The method attempts to extract the namespace from the name.
2. The method attempts to extract the namespace from a previously declared namespace.
3. The method attempts to generate the default namespace.

If *name* contains a prefix and *namespace-URI* is used, and this call is the first instance of the *namespace-URI,* then the namespace will be added to the element. This is equivalent to calling the DECLARE-NAMESPACE method. If only the *name* is used and it contains a prefix, then the SAX-writer attempts to resolve the prefix to a namespace.

This technique is logically equivalent to calling the START-ELEMENT, WRITE-CHARACTERS, and END-ELEMENT methods where *name* and *namespace-URI* are the parameters of START-ELEMENT and END-ELEMENT, and *chardata* is the parameter of WRITE-CHARACTERS.

Note that attributes and namespaces cannot be added after you call this method. If you need to add either to the element, then use the START-ELEMENT method.

If the STRICT attribute is TRUE, the FRAGMENT attribute is FALSE, and the invocation would result in more than one document-level element (that is, root node), then the method fails. Also, if STRICT is TRUE, an external DTD has been declared, and the invocation would create the root node, the *name* used for the DTD declaration must match the *name* of the root node or the method fails.

Note: The WRITE-DATA-ELEMENT( ) method allows some non-printing characters to be written to XML documents. These characters (for example, #x1, #x1F) are not part of the XML Version 1.0 standard, but are allowed in order to support ABL applications.

See also

DECLARE-NAMESPACE( ) method

WRITE-EMPTY-ELEMENT( ) method

Creates an empty XML node in a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-EMPTY-ELEMENT ( name [ , namespace-URI ]
  [ , SAX-attributes-handle ] )
```

name

A CHARACTER or LONGCHAR expression evaluating to the fully qualified or unqualified name of the element.

namespace-URI

A CHARACTER or LONGCHAR expression evaluating to the URI of the element, or an empty string ("") or the Unknown value (?) if the element doesn't contain a namespace.

SAX-attributes-handle

A HANDLE to a SAX-attributes object. The attributes in the SAX-attributes object are added to the new empty XML element.

Creates an empty XML node. This method call sets the WRITE-STATUS to SAX-WRITE-TAG.

If *namespace-URI* is present, then the prefix is resolved in the following order:

1. The method attempts to extract the namespace from the *name*.
2. The method attempts to extract the namespace from a previously declared namespace.
3. The method attempts to generate the default namespace.

If the *name* contains a prefix, *namespace-URI* is present, and this is the first instance of the *namespace-URI* then the namespace is added to the element. This is equivalent to calling the DECLARE-NAMESPACE method.

If only the *name* is present and it contains a prefix, then the SAX-writer attempts to resolve the prefix to a namespace.

Although this method call appears to be logically equivalent to a START-ELEMENT invocation directly followed by its corresponding END-ELEMENT invocation, the two techniques produce different outputs. The START-ELEMENT and END-ELEMENT methods produce a pair of tags; `<element></element>`. WRITE-EMPTY-ELEMENT produces the empty element tag; `<element/>`.

If the *SAX-attributes-handle* is supplied, the attributes will be written after the name of the element, in the order that they are indexed in the SAX-attributes object. If they contain namespace information, then that information will be handled as if the attribute were added using INSERT-ATTRIBUTE( ) method.

If STRICT is TRUE, FRAGMENT is FALSE, and the invocation would result in more than one document-level element, (that is, root node), then the method fails.

WRITE-ENTITY-REF( ) method

Adds an entity reference to the XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-ENTITY-REF ( value )
```

value

A CHARACTER or LONGCHAR expression evaluating to the value of the entity reference.

Call this method to add an entity reference to the XML document. You can add entity references at any time during the write.

This method does not change the WRITE-STATUS attribute.

You cannot add entity references using the WRITE-CHARACTERS method because the entity references contain the escapable character `"&"`. For example, if you add the entity reference `fromname` with the WRITE-CHARACTERS method, this call, `xmlwh:WRITE-CHARACTERS("&fromname;")`, produces the invalid value `&fromname;` in the XML document. However, you can add entity references using the WRITE-FRAGMENT method, since it does not escape special characters.

Do not include the special characters when inserting the reference, only the reference value. For example, `xmlwh:entity-reference("fromname")` produces `&fromname;` in the XML document.

See also

WRITE-CHARACTERS( ) method, WRITE-FRAGMENT( ) method

WRITE-EXTERNAL-DTD( ) method

Adds an external Document Type Definition (DTD) reference to an XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-EXTERNAL-DTD ( name , systemID [ , publicID ] )
```

name

A CHARACTER or LONGCHAR expression evaluating to the fully qualified or unqualified name of the XML document root node.

systemID

A CHARACTER or LONGCHAR expression evaluating to the system ID of the DTD.

publicID

A CHARACTER or LONGCHAR expression evaluating to the public ID of the DTD.

Call this method to add an external DTD reference to the prolog of the XML document.

You can only call this method before the first call of START-ELEMENT. That is, only call this method when the WRITE-STATUS is SAX-WRITE-BEGIN. After the call, the status remains SAX-WRITE-BEGIN.

If the STRICT attribute is TRUE, and you call this method after you create the root element (or when the CREATE-FRAGMENT attribute is TRUE), then the method fails.

The value of *name* must match the value in the root node. If the STRICT attribute is TRUE and the two values do not match, then the method fails and generates an error.

WRITE-FRAGMENT( ) method

Adds character data to the XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-FRAGMENT ( { chardata | longchar | x-noderef } )
```

chardata

An expression that evaluates to a CHARACTER variable that contains the XML text.

longchar

An expression that evaluates to a LONGCHAR variable that contains the XML text.

noderef

A valid X-NODEREF handle that contains the XML text.

Call this method to add un-escaped CHARACTER data to the XML document. This allows the adding of XML fragments to the document without the special characters being escaped to their XML representation. For example, "<" escapes to <. It is up to the developer to ensure that the characters written are proper XML with the correct characters escaped. Even if the STRICT attribute is TRUE, the SAX-writer will not validate what is written.

You can call this method at any time during the write. This method changes the WRITE-STATUS attribute to SAX-WRITE-CONTENT.

Note: The WRITE-FRAGMENT( ) method allows some non-printing characters to be written to XML documents. These characters (for example, #x1, #x1F) are not part of the XML Version 1.0 standard, but are allowed in order to support ABL applications.

WRITE-JSON( ) method

Writes a specified JSON string, `Progress.Json.ObjectModel.JsonArray` object, or `Progress.Json.ObjectModel.JsonObject` object from a corresponding ProDataSet, temp-table, or temp-table buffer object. In the case of a temp-table buffer object, all the records of the temp-table associated with the buffer object are written to the JSON string or `JsonArray` object—not just the one record in the buffer.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

Syntax

```
WRITE-JSON ( target-type
  , { file | stream | stream-handle | memptr | longchar
        | JsonArray | JsonObject }
  [ , formatted [ , encoding [ , omit-initial-values
  [ , omit-outer-object [ , write-before-image ] ] ] ] ] )
```

target-type

A CHARACTER expression that specifies the target type for the JSON output. Valid values are "FILE", "STREAM", "STREAM-HANDLE", "MEMPTR", "LONGCHAR", "JsonArray", and "JsonObject".

All values except "JsonArray" and "JsonObject" specify a target for a JSON string.

"JsonArray" is only a valid *target-type* for invoking WRITE-JSON( ) on a temp-table or buffer object handle. If set to "JsonArray" for a ProDataSet object handle, the WRITE-JSON( ) method generates an error message and returns a value of FALSE.

file

A CHARACTER expression that specifies the name of a file to which the AVM writes the JSON string. You can specify an absolute pathname or a pathname relative to the current working directory. If a file with the specified name already exists, the AVM verifies that the file is writeable and overwrites the file.

stream

A CHARACTER expression that specifies the name of a stream. If you specify the empty string (""), the AVM writes the JSON string to the default unnamed output stream. For WebSpeed, write the JSON string to the WebSpeed-defined output stream (WEBSTREAM).

For more information about using ABL unnamed output streams, see the DEFINE STREAM statement reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces*.

stream-handle

A HANDLE variable that specifies a stream object handle.

memptr

A MEMPTR variable to contain the JSON string in memory. If you do not specify the encoding parameter, the AVM encodes the text written to the MEMPTR as UTF-8. This method allocates the required amount of memory for the JSON string and sets the size of the variable. When you are finished using the MEMPTR, you must free the associated memory, by executing SET-SIZE(*memptr*) = 0 on the MEMPTR.

longchar

A LONGCHAR variable to contain the JSON string in memory.

The AVM saves the JSON string to the LONGCHAR variable in the code page that corresponds to the character encoding you specify in the *encoding* option. If you do not specify a character encoding for the JSON string, the AVM saves the LONGCHAR variable in UTF-8.

If the LONGCHAR variable's code page is fixed (that is, set using the FIX-CODEPAGE statement) and the fixed code page is not equivalent to the character encoding you specify in the *encoding* option, the WRITE-JSON( ) method generates an error message and returns FALSE. The JSON string is not saved to the LONGCHAR.

JsonArray

A reference to a Progress.Json.ObjectModel.JsonArray object to which the AVM writes the contents of the associated temp-table. Any prior contents of the specified *JsonArray* are removed.

The *JsonArray* must be a valid instance of type Progress.Json.ObjectModel.JsonArray. If not, the WRITE-JSON( ) method generates an error message and returns FALSE.

JsonObject

A reference to a Progress.Json.ObjectModel.JsonObject object to which the AVM writes the contents of the associated ProDataSet or temp-table. Any prior contents of the specified *JsonObject* are removed.

The *JsonObject* must be a valid instance of type Progress.Json.ObjectModel.JsonObject. If not, the WRITE-JSON( ) method generates an error message and returns FALSE.

formatted

An optional LOGICAL expression where TRUE directs the AVM to format the JSON string in a hierarchical manner using extra white space, carriage returns, and line feeds. The default value is FALSE.

If you specify the Unknown value (?), the method uses the default value of FALSE.

The *formatted* option is ignored for *JsonArray* and *JsonObject* targets.

encoding

An optional CHARACTER expression that specifies the name of the character encoding the AVM uses to write the JSON output. The default encoding is "UTF-8".

The encoding name must specify a Unicode transformation format. Valid values are "UTF-8", "UTF-16", "UTF-16BE", "UTF-16LE", "UTF-32", "UTF-32BE", and "UTF-32LE".

Note: If you specify the empty string ("") or the Unknown value (?), the AVM uses the default encoding of UTF-8.

omit-initial-values

An optional LOGICAL expression where TRUE directs the AVM to exclude temp-table fields containing their initial values from the JSON output, and FALSE directs the AVM to include all temp-table field data in the output. The default value is FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

When working with large ProDataSets, omitting fields containing their initial values can yield smaller JSON values, more efficient network transfers, and performance gains with the READ-JSON( ) and WRITE-JSON( ) methods.

This behavior applies both to temp-table fields that have the default initial value for its data type, and for fields that have an initial value set with the ABL INITIAL option.

Although using the *omit-initial-values* option can give your application performance and resource use improvements, you must be sure that the consumers of the generated JSON output can correctly handle it. The ABL READ-JSON( ) method always populates created records with initial values from the temp-table or ProDataSet definition. Other applications might not do this.

omit-outer-object

An optional LOGICAL expression that indicates whether the outer-most object is included in the JSON output. TRUE directs the AVM to remove the outer-most object on output. FALSE directs the AVM to include the outer-most object in the output.

For a JSON string or *JsonObject* target, the default value is FALSE. If you specify the Unknown value (?), the method also uses the default value of FALSE. When writing a temp-table to a *JsonObject* target, *omit-outer-object* **must** be FALSE. If you specify TRUE, the WRITE-JSON( ) method generates an error message and returns FALSE.

For a *JsonArray* target, the default value for *omit-outer-object* is TRUE. If you specify the Unknown value (?), the method also uses the default value of TRUE. When writing a temp-table or temp-table buffer object to a *JsonArray* target, *omit-outer-object* **must** be TRUE. If you specify FALSE, the WRITE-JSON( ) method generates an error message and returns FALSE.

write-before-image

An optional LOGICAL expression where TRUE directs the AVM to include ProDataSet before-image data and error information in the JSON output. The default value is FALSE.

This element can only be set to TRUE for a ProDataSet. If its value is set to TRUE for a temp-table or buffer handle, the WRITE-JSON( ) method generates an error message and returns a value of FALSE.

Notes

- When writing data from a ProDataSet object that contains data-relations, you can nest child rows of a ProDataSet buffer within their parent rows in the resulting JSON output by:

  - Setting the NESTED attribute on the data-relation object handle to TRUE
  - Specifying the NESTED option for the data-relation on the DEFINE DATASET statement
  - Specifying the NESTED option in the ADD-RELATION( ) method
  - Specify a PARENT-ID-RELATION in the DEFINE DATASET statement
  - Using the ADD-PARENT-ID-RELATION( ) method on a dynamic ProDataSet object

- You cannot call WRITE-JSON( ) on a database buffer handle.
- WRITE-JSON( ) can serialize fields with data types based on class-based objects. The following restrictions apply to any class-based objects that are to be written to the JSON target:
  - The object's class and all of the classes in its hierarchy must be marked as SERIALIZABLE. See the CLASS statement entry for more information.
  - All of the object's data members that are themselves defined as class-based objects must be of class types that are also marked SERIALIZABLE. ABL enums are also allowed because they are implicitly serializable.
  - By default, all non-static, public data members and read-write properties are serialized. You can include protected and private data members, read-write properties, temp tables, and ProDataSets in serialization by defining them using the SERIALIZABLE option. See the DEFINE entry for each element for more details.
  - To be able to deserialize an object, the API for the object's class at the time of serialization must be compatible with the API for the class at the time of deserialization. At a minimum, the data members contained in the JSON target for that object should exist in the class, be serializable, and have compatible data types.
  - The content of static data members is not serialized, and the state of queries, buffers, open files, streams, and event subscriptions, for example, are not maintained.
- When executing the method, there is no guarantee as to the contents of the buffers involved after the method finishes executing.
- Binary types, including BLOB, RAW, and ROWID are base64-encoded/decoded. See Data type mapping in *Use JSON with ABL Applications* for more information.

Examples

The following code example defines a static ProDataSet object, attaches its data sources, fills the ProDataSet object, and writes the ProDataSet object to a JSON string in a nested manner:

```
DEFINE VARIABLE cTargetType AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFile       AS CHARACTER NO-UNDO.
DEFINE VARIABLE lFormatted  AS LOGICAL   NO-UNDO.
DEFINE VARIABLE lRetOK      AS LOGICAL   NO-UNDO.

DEFINE TEMP-TABLE ttCustomer NO-UNDO LIKE Customer.
DEFINE TEMP-TABLE ttOrder    NO-UNDO LIKE Order.
DEFINE TEMP-TABLE ttInvoice  NO-UNDO LIKE Invoice.

DEFINE DATASET dsOrderLog FOR ttCustomer, ttOrder, ttInvoice
  DATA-RELATION CustOrd FOR ttCustomer,
    ttOrder RELATION-FIELDS(CustNum,CustNum) NESTED
  DATA-RELATION OrdInv FOR ttOrder,
    ttInvoice RELATION-FIELDS(OrderNum,OrderNum) NESTED.

DEFINE DATA-SOURCE dsCustomer FOR Customer.
DEFINE DATA-SOURCE dsOrder    FOR Order.
DEFINE DATA-SOURCE dsInvoice  FOR Invoice.

BUFFER ttCustomer:HANDLE:ATTACH-DATA-SOURCE(DATA-SOURCE dsCustomer:HANDLE).
BUFFER ttOrder:HANDLE:ATTACH-DATA-SOURCE(DATA-SOURCE dsOrder:HANDLE).
BUFFER ttInvoice:HANDLE:ATTACH-DATA-SOURCE(DATA-SOURCE dsInvoice:HANDLE).

DATA-SOURCE dsCustomer:FILL-WHERE-STRING = "WHERE Customer.CustNum = 2 ".
DATASET dsOrderLog:FILL().

ASSIGN
  cTargetType = "file"
  cFile       = "dset.json"
  lFormatted  = TRUE.
lRetOK = DATASET dsOrderLog:WRITE-JSON(cTargetType, cFile, lFormatted).
```

The following code example defines a static temp-table object, populates the temp-table object (code not shown), and writes the temp-table object to a JSON string:

```
DEFINE VARIABLE cTargetType AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFile       AS CHARACTER NO-UNDO.
DEFINE VARIABLE lFormatted  AS LOGICAL   NO-UNDO.
DEFINE VARIABLE lRetOK      AS LOGICAL   NO-UNDO.

DEFINE TEMP-TABLE ttCust NO-UNDO LIKE Customer.

/* Code to populate the temp-table */
ASSIGN
  cTargetType = "file"
  cFile       = "ttCust.json"
  lFormatted  = TRUE.
lRetOK = TEMP-TABLE ttCust:WRITE-JSON(cTargetType, cFile, lFormatted).
```

The following code example defines and populates a static temp-table object with different assigned, initial, and default value settings. It also defines and instantiates both a `JsonArray` and `JsonObject` object, then writes the JSON representation of the temp-table to both the `JsonArray` and the `JsonObject`. As it writes out the JSON, it generates `MESSAGE` statement output to a `TtabWriteJsonArrObj.out` file to show the results by returning and displaying both the written JSON array and object contents as a string.

Table 111: Procedure writing JSON from a temp-table to a JsonArray and JsonObject instance

```
USING Progress.Json.ObjectModel.*.

DEFINE VARIABLE jArr AS JsonArray.
DEFINE VARIABLE jObj AS JsonObject.
DEFINE VARIABLE htt AS HANDLE.

DEFINE TEMP-TABLE ttTest
    FIELD f1 AS CHARACTER INITIAL "InitVal"
    FIELD f2 AS INTEGER.

OUTPUT TO TtabWriteJsonArrObj.out.

CREATE ttTest.
ASSIGN
    f1 = "AAA"
    f2 = 1.

CREATE ttTest.
ASSIGN
    f2 = 2.

CREATE ttTest.
ASSIGN
    f1 = "BBB".

htt = TEMP-TABLE ttTest:HANDLE.

jArr = NEW JsonArray().
jObj = NEW JsonObject().

MESSAGE "ttTest:WRITE-JSON to jArr".

htt:WRITE-JSON("JsonArray", jArr).

MESSAGE "After calling WRITE-JSON() JsonArray contains:".

MESSAGE STRING(jArr:GetJsonText()).
MESSAGE "jArr:LENGTH =" jArr:LENGTH.

MESSAGE SKIP "ttTest:WRITE-JSON to jObj".

htt:WRITE-JSON("JsonObject", jObj).

MESSAGE "After calling WRITE-JSON () JsonObject contains:" SKIP.
MESSAGE STRING(jObj:GetJsonText()).

OUTPUT CLOSE.
```

The following shows the MESSAGE statement output generated in `TtabWriteJsonArrObj.out`.

Table 112: MESSAGE statement output written to the example `TtabWriteJsonArrObj.out` file

```
ttTest:WRITE-JSON to jArr
After calling WRITE-JSON() JsonArray contains:
[{"f1":"AAA","f2":1},{"f1":"InitVal","f2":2},{"f1":"BBB","f2":0}]

jArr:LENGTH = 3

ttTest:WRITE-JSON to jObj
After calling WRITE-JSON () JsonObject contains:
{"ttTest":[{"f1":"AAA","f2":1},{"f1":"InitVal","f2":2},{"f1":"BBB","f2":0}]}
```

See also

ENCODING attribute, FIX-CODEPAGE statement, FOREIGN-KEY-HIDDEN attribute, NESTED attribute, Progress.Json.ObjectModel.JsonArray class, Progress.Json.ObjectModel.JsonObject class, READ-JSON( ) method, SERIALIZE-ROW( ) method

WRITE-MESSAGE( ) method

Writes a user message to the current log file.

For an interactive or batch client, the WRITE-MESSAGE( ) method writes the log entries to the log file specified by the LOGFILE-NAME attribute or the Client Logging (`-clientlog`) startup parameter. For application servers, the WRITE-MESSAGE( ) method writes the log entries to the server log file. For DataServers, the WRITE-MESSAGE( ) method writes the log entries to the log file specified by the DataServer Logging (`-dslog`) startup parameter.

Note: The DSLOG-MANAGER's LOGFILE-NAME attribute is read-only.

Return type: LOGICAL

Applies to: DSLOG-MANAGER system handle, LOG-MANAGER system handle

Syntax

```
WRITE-MESSAGE ( msg-exp [ , subsys-expr ] )
```

msg-exp

A character expression or variable representing the message to write to the log file.

subsys-expr

A character expression representing the subsystem identifier to write to the log file. The default is `"APPL"`. You can provide your own subsystem identifier. The subsystem identifier has a character limit 10 characters, and is padded to 10 characters. If you provide a subsystem identifier longer than 10 characters, WRITE-MESSAGE( ) writes only the first 10 characters.

If the WRITE-MESSAGE( ) method succeeds, it returns TRUE. If it fails, it returns FALSE.

If there is no client log file, the WRITE-MESSAGE( ) method returns FALSE and displays a warning message indicating this operation is not valid when there is no log file. For an interactive or batch client, the WRITE-MESSAGE( ) method writes the warning message to the current output device.

When the client writes messages using the WRITE-MESSAGE( ) method, the component identifier in the message header is the default component identifier for the client executable writing to the log. For example, the component identifier for a GUI or character client is `"4GL"`. You can provide your own subsystem identifier. The default is `"APPL"`. The subsystem identifier has a character limit of 10 characters, and is padded to 10 characters.

The following is an example:

```
LOG-MANAGER:WRITE-MESSAGE("Got here, x=" + STRING(x), "DEBUG1").
```

The following line appears in the log file:

```
[04/12/05@13:19:19.742-0500] P-003616 T-001984 1 4GL DEBUG1    Got here, x=5
```

WRITE-PROCESSING-INSTRUCTION( ) method

Creates a processing instruction node in an XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-PROCESSING-INSTRUCTION ( target, data )
```

target

A CHARACTER or LONGCHAR expression evaluating to the target of the processing instruction.

data

A CHARACTER or LONGCHAR expression that evaluates to the data associated with the processing instruction.

Call this method to add a processing instruction node to the XML document. You can add processing instructions at any time. The SAX-writer object creates the processing instruction by enclosing the CHARACTER expression in open and close processing instruction markers (<? and ?>). For example:

```
<?xml version="1.0"?>
```

This method does not change the WRITE-STATUS attribute.

WRITE-STATUS attribute

The current state of a XML write in a SAX-writer object.

Data type: INTEGER

Access: Read-only

Applies to: SAX-writer object handle

The default value is SAX-WRITE-IDLE.

The possible values WRITE-STATUS can assume are shown in the following table.

Table 113: WRITE-STATUS attribute values

WRITE-STATUS	Description	Integer value
SAX-WRITE-IDLE	No writing has occurred.	1
SAX-WRITE-BEGIN	The START-DOCUMENT method has been called and writing has begun.	2
SAX-WRITE-TAG	The writer has written an opening tag. This is the only time that attributes can be inserted with INSERT-ATTRIBUTE and DECLARE-NAMESPACE.	3
SAX-WRITE-ELEMENT	The writer is within an element.	4
SAX-WRITE-CONTENT	The writer has written the content of an element. In other words, the WRITE-CHARACTERS method has been called.	5
SAX-WRITE-COMPLETE	The END-DOCUMENT method has been called and writing is complete.	6
SAX-WRITE-ERROR	The SAX-writer could not start or could not continue. Likely causes include: SAX-writer could not be loaded, the XML target could not be written to, a method call fails, etc. This is the status if there is an invalid XML generated while STRICT is TRUE. If the status is SAX-WRITE-ERROR then no attributes can be written and the only method that can be called is RESET.	7

WRITE-XML( ) method

Writes an XML document from a ProDataSet, temp-table, or temp-table buffer object. In the case of the temp-table buffer object, all the records of the temp-table associated with the buffer object are written to the XML document—not just the one in the buffer. You can write the XML representation of the object with data, schema, or both. If you include schema, it is written using the XML Schema Definition (XSD) language.

When writing data from a ProDataSet object, the AVM writes the current version of data in each row of each table in the ProDataSet object. However, you can also include any before-image data, so that both the current and original versions of the data in each table row are written.

When writing schema for a ProDataSet object, the AVM writes all table definitions as well as relation and index definitions. When writing schema for a temp-table or temp-table buffer object, the AVM writes only table and index definitions.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

Syntax

```
WRITE-XML ( target-type
  , { file | stream | stream-handle | memptr | handle | longchar }
  [ , formatted [ , encoding [ , schema-location [ , write-xmlschema
  [ , min-xmlschema [ , write-before-image [ , omit-initial-values
  ] ] ] ] ] ] ] )
```

target-type

A CHARACTER expression that specifies the target XML document type. Valid values are: "FILE", "STREAM", "STREAM-HANDLE", "MEMPTR", "HANDLE", and "LONGCHAR".

file

A CHARACTER expression that specifies the name of a file to which the AVM writes the XML document text. You can specify an absolute pathname or a relative pathname (based on the current working directory). If a file with the specified name already exists, the AVM verifies that the file is writeable and overwrites the file. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

stream

A CHARACTER expression that specifies the name of a stream. If you specify the empty string (""), the AVM writes the XML document text to the default unnamed output stream. For WebSpeed, write the XML document text to the WebSpeed-defined output stream (WEBSTREAM).

For more information about using ABL unnamed output streams, see the DEFINE STREAM statement reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces*.

stream-handle

A HANDLE variable that specifies a stream object handle.

memptr

A MEMPTR variable to contain the XML document text in memory. The method allocates the required amount of memory for the XML document text and sets the size of the variable. When you are finished using the MEMPTR, you must free the associated memory by executing SET-SIZE(*memptr*) = 0 on the MEMPTR.

handle

An X-document object handle or X-noderef object handle. If the specified handle contains XML text, the AVM deletes the existing text first.

longchar

A LONGCHAR variable to contain the XML document text in memory.

The AVM saves the XML document text to the LONGCHAR variable in the code page that corresponds to the character encoding you specify in the *encoding* option. If you do not specify a character encoding for the XML document text, the AVM saves the LONGCHAR variable in UTF-8.

If the LONGCHAR variable's code page is fixed (that is, set using the FIX-CODEPAGE statement) and the fixed code page is not equivalent to the character encoding you specify in the *encoding* option, the WRITE-XML( ) method returns an error and the XML document is not saved to the LONGCHAR.

formatted

An optional LOGICAL expression where TRUE directs the AVM to format the XML document text in a hierarchical manner using extra white space, carriage returns, and line feeds. The default value is FALSE.

If you specify the Unknown value (?), the method uses the default value of FALSE.

encoding

An optional CHARACTER expression that specifies the name of the character encoding the AVM uses to write the XML document text. The default encoding is UTF-8.

The encoding name must be an Internet Assigned Numbers Authority (IANA) name supported by the ABL XML Parser. For a list of supported IANA encodings and their corresponding ABL code pages, see Table 77: IANA encodings and corresponding ABL code pages in the ENCODING attribute reference entry.

Note: The AVM records this character encoding in the encoding declaration in the XML document's prologue. If you specify the empty string ("") or the Unknown value (?), the AVM uses the default encoding of UTF-8. In this case, the AVM does not record the character encoding in the XML document's encoding declaration.

If *target-type* is "HANDLE", the X-document's ENCODING attribute is also set.

schema-location

An optional CHARACTER expression that specifies the name of an external XML Schema file. The method uses this value to set the xsi:schemaLocation or xsi:noNamespaceSchemaLocation attribute in the XML document. If the ProDataSet or temp-table object's NAMESPACE-URI attribute is the empty string ("") or the Unknown value (?), the method adds the xsi:noNamespaceSchemaLocation attribute in the XML document and sets it to this value. If the NAMESPACE-URI attribute is not the empty string ("") or the Unknown value (?), the method adds the xsi:schemaLocation attribute to the XML document with a value of "namespace-uri<space>*schema-location*". The default value is the Unknown value (?). The filename can contain Unicode characters.

Note: You must provide the location of an actual XML Schema file. Consider using the WRITE-XMLSCHEMA( ) method to generate the XML Schema file.

If you specify *write-xmlschema* as TRUE, you cannot specify *schema-location*.

write-xmlschema

An optional LOGICAL expression where TRUE directs the AVM to write the ProDataSet or temp-table object's relational structure as in-line XML Schema along with the data, and FALSE directs the AVM to write only the data. The default value is FALSE.

If you specify TRUE, you cannot specify *schema-location*. If you specify FALSE, you must also specify *min-xmlschema* as FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

Note: If you specify TRUE and the NAMESPACE-URI attribute value for a temp-table buffer within a ProDataSet object is different than that of the ProDataSet object, the method creates a separate XML Schema file for the temp-table definition. The namespace URI for the temp-table is imported into the ProDataSet schema, with a schemaLocation pointing to a separate XML Schema file containing the temp-table definition. Multiple namespaces are supported only when *target-type* is "FILE". If the ProDataSet object contains multiple namespaces and *target-type* is not "FILE", the method generates an error and returns FALSE.

min-xmlschema

An optional LOGICAL expression where TRUE directs the AVM to write the minimum amount of schema when it writes the XML Schema representation of the object, and FALSE directs the AVM to write the complete schema including ABL-specific schema attributes. The default value is FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

When TRUE, ABL-specific schema information (such as, field format, non-unique indexes, and so on) is omitted from the XML Schema. If the ABL data type of the temp-table field is not the default ABL data type for the XML Schema type, the AVM writes the prodata:*dataType* XML Schema attribute for the field. If the initial value of the temp-table field is TODAY, NOW, or UNKNOWN (and UNKNOWN is not the default initial value for the field's data type), the AVM writes the prodata:*initial* XML Schema attribute for the field.

When TRUE, the XML Schema will contain any ABL-specific XML Schema attributes needed to define the data relations for a ProDataSet.

If you specify *write-xmlschema* as FALSE, you must also specify *min-xmlschema* as FALSE.

write-before-image

An optional LOGICAL expression where TRUE directs the AVM to write any before-image table data and error information in addition to the ProDataSet object data, and FALSE directs the AVM to write only the ProDataSet object data. The default value is FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

omit-initial-values

An optional LOGICAL expression where TRUE directs the AVM to exclude temp-table fields containing their initial values from the XML document, and FALSE directs the AVM to include all temp-table field data in the XML. The default value is FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

When working with large ProDataSets, omitting initial values in records can yield smaller XML documents, more efficient network transfers, and performance gains with the READ-XML( ) and WRITE-XML( ) methods.

The XML Schema indicates what the initial values for the omitted temp-table fields are, thus ensuring round trip of the XML data from and to OpenEdge applications. The READ-XML( ) method assigns the field its initial value when the field is not present in the XML document.

This behavior applies both to temp-table fields that have the default initial value for its data type and for fields that have an initial value set with the ABL INITIAL option.

Although using the omit-initial-values option can give your application performance and resource use improvements, you need to be sure that the consumers of the generated XML document will correctly handle the XML. The ABL READ-XML( ) method will always populate created records with initial values from the temp-table or ProDataSet definition. Other applications might not do this.

For example, .NET can read an XML Schema and XML data document into a dynamic ADO .NET DataSet using its `ReadXmlSchema( )` and `ReadXml( )` APIs, but missing elements are always interpreted as a null (the Unknown value (?) in ABL terms) DataColumn in the DataTable. They do recognize the initial value definition in the XML Schema when creating rows in the DataTable directly (initial value in the XML Schema gets translated to the `DefaultValue` property on the DataColumn), but do not with `ReadXml( )`.

Note: For an array field to be omitted, each element of the array must contain the initial value.

Notes

- You can specify how a temp-table column is represented in XML (that is, as an ELEMENT, ATTRIBUTE, or TEXT) by:
  - Setting the `XML-NODE-TYPE` attribute on the Buffer-field object handle
  - Specifying the `XML-NODE-TYPE` option on the `DEFINE TEMP-TABLE` statement
- When writing data from a ProDataSet object that contains data-relations, you can nest child rows of a ProDataSet buffer within their parent rows in the resulting XML document by:
  - Setting the `NESTED` attribute on the data-relation object handle to TRUE
  - Specifying the `NESTED` option for the data-relation on the `DEFINE DATASET` statement
  - Specifying the `NESTED` option in the ADD-RELATION( ) method
  - Specify a `PARENT-ID-RELATION` in the DEFINE DATASET statement
  - Using the ADD-PARENT-ID-RELATION( ) method on a dynamic ProDataSet object
- ABL has pairs of attributes with overlapping purposes in how the AVM outputs XML data:
  - `SERIALIZE-HIDDEN` and `XML-NODE-TYPE`
  - `SERIALIZE-NAME` and `XML-NODE-NAME`
- Refer to the attribute entries for a full description of their interaction.
- If your temp-tables contain array fields, third party products utilizing the XML might not map the ABL array field to an array column or object. For best interoperability with third party products, flatten array fields into individual fields.
- You cannot write an XML document from a database buffer.
- When executing the method, there is no guarantee as to the contents of the buffers involved after the method finishes executing.
- The `WRITE-XML( )` method allows some non-printing characters to be written to XML documents. These characters (for example, #x1, #x1F) are not part of the XML Version 1.0 standard, but are allowed in order to support ABL applications.
- Binary types, including BLOB, RAW, and ROWID are base64-encoded/decoded, unless the XML-DATA-TYPE attribute is hexBinary. If `XML-DATA-TYPE` is hexBinary, then BLOB, RAW, and ROWID are hex-encoded/decoded. See Default mappings from other XML Schema data types to ABL data types in *Use XML with ABL Applications* for more information.

Examples

The following code example defines a static ProDataSet object, attaches its data sources, fills the ProDataSet object, and writes the ProDataSet object to an XML document in a nested manner:

```
DEFINE VARIABLE cTargetType     AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFile           AS CHARACTER NO-UNDO.
DEFINE VARIABLE lFormatted      AS LOGICAL   NO-UNDO.
DEFINE VARIABLE cEncoding       AS CHARACTER NO-UNDO.
DEFINE VARIABLE cSchemaLocation AS CHARACTER NO-UNDO.
DEFINE VARIABLE lWriteSchema    AS LOGICAL   NO-UNDO.
DEFINE VARIABLE lMinSchema      AS LOGICAL   NO-UNDO.
DEFINE VARIABLE lRetOK          AS LOGICAL   NO-UNDO.

DEFINE TEMP-TABLE ttCustomer NO-UNDO LIKE Customer.
DEFINE TEMP-TABLE ttOrder    NO-UNDO LIKE Order.
DEFINE TEMP-TABLE ttInvoice  NO-UNDO LIKE Invoice.

DEFINE DATASET DSET FOR ttCustomer, ttOrder, ttInvoice
  DATA-RELATION CustOrd FOR ttCustomer,
    ttOrd RELATION-FIELDS(CustNum,CustNum) NESTED
  DATA-RELATION OrdInv FOR ttOrder,
    ttInv RELATION-FIELDS(OrderNum,OrderNum) NESTED.

DEFINE DATA-SOURCE dsCustomer FOR Customer.
DEFINE DATA-SOURCE dsOrder    FOR Order.
DEFINE DATA-SOURCE dsInvoice  FOR Invoice.

BUFFER ttCustomer:HANDLE:ATTACH-DATA-SOURCE(DATA-SOURCE dsCustomer:HANDLE).
BUFFER ttOrder:HANDLE:ATTACH-DATA-SOURCE(DATA-SOURCE dsOrder:HANDLE).
BUFFER ttInvoice:HANDLE:ATTACH-DATA-SOURCE(DATA-SOURCE dsInvoice:HANDLE).

DATA-SOURCE dsCustomer:FILL-WHERE-STRING = "WHERE Customer.CustNum = 2 ".
DATASET DSET:FILL().

ASSIGN
  cTargetType     = "file"
  cFile           = "dset.xml"
  lFormatted      = TRUE
  cEncoding       = ?
  cSchemaLocation = ?
  lWriteSchema    = FALSE
  lMinSchema      = FALSE.

lRetOK = DATASET DSET:WRITE-XML(cTargetType, cFile, lFormatted, cEncoding,
  cSchemaLocation, lWriteSchema, lMinSchema).
```

The following code example defines a static temp-table object, populates the temp-table object (code not shown), and writes the temp-table object to an XML document:

```
DEFINE VARIABLE cTargetType     AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFile           AS CHARACTER NO-UNDO.
DEFINE VARIABLE lFormatted      AS LOGICAL   NO-UNDO.
DEFINE VARIABLE cEncoding       AS CHARACTER NO-UNDO.
DEFINE VARIABLE cSchemaLocation AS CHARACTER NO-UNDO.
DEFINE VARIABLE lWriteSchema    AS LOGICAL   NO-UNDO.
DEFINE VARIABLE lMinSchema      AS LOGICAL   NO-UNDO.
DEFINE VARIABLE lRetOK          AS LOGICAL   NO-UNDO.

DEFINE TEMP-TABLE ttCust NO-UNDO LIKE Customer.

/* Code to populate the temp-table  */
ASSIGN
  cTargetType     = "file"
  cFile           = "ttCust.xml"
  lFormatted      = TRUE
  cEncoding       = ?
  cSchemaLocation = ?
  lWriteSchema    = FALSE
  lMinSchema      = FALSE.

lRetOK = TEMP-TABLE ttCust:WRITE-XML(cTargetType, cFile,lFormatted, cEncoding,
  cSchemaLocation, lWriteSchema, lMinSchema).
```

See also

ENCODING attribute, FIX-CODEPAGE statement, FOREIGN-KEY-HIDDEN attribute, NAMESPACE-PREFIX attribute, NAMESPACE-URI attribute, NESTED attribute, READ-XML( ) method, READ-XMLSCHEMA( ) method, SERIALIZE-ROW( ) method, SERIALIZE-HIDDEN attribute, SERIALIZE-NAME attribute, SERIALIZE-ROW( ) method, WRITE-XMLSCHEMA( ) method, XML-NODE-NAME attribute, XML-NODE-TYPE attribute

WRITE-XMLSCHEMA( ) method

Writes an XML representation of the definition of a ProDataSet, temp-table, or temp-table buffer object (that is, an XML Schema file). The XML Schema is written using the XML Schema Definition (XSD) language.

When writing schema for a ProDataSet object, the AVM writes all table definitions as well as relation and index definitions. When writing schema for a temp-table or temp-table buffer object, the AVM writes only table and index definitions.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

Syntax

```
WRITE-XMLSCHEMA ( target-type
  , { file | stream | stream-handle | memptr | handle | longchar }
  [ , formatted [ , encoding [ , min-xmlschema [ , omit-initial-values
  ] ] ] ] )
```

target-type

A CHARACTER expression that specifies the target XML Schema document type. Valid values are: "FILE", "STREAM", "STREAM-HANDLE", "MEMPTR", "HANDLE", and "LONGCHAR".

file

A CHARACTER expression that specifies the name of a file to which the AVM writes the XML Schema document text. You can specify an absolute pathname or a relative pathname (based on the current working directory). If a file with the specified name already exists, the AVM verifies that the file is writeable and overwrites the file. The pathname can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode.

stream

A CHARACTER expression that specifies the name of a stream. If you specify the empty string (""), the AVM writes the XML Schema document text to the default unnamed output stream. For WebSpeed, write the XML Schema document text to the WebSpeed-defined output stream (WEBSTREAM).

For more information about using ABL unnamed output streams, see the DEFINE STREAM statement reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces*.

stream-handle

A CHARACTER expression that specifies a stream object handle.

memptr

A MEMPTR variable to contain the XML Schema document text in memory. The method allocates the required amount of memory for the XML document text and sets the size of the variable. When you are finished using the MEMPTR, you must free the associated memory by executing SET-SIZE(*memptr*) = 0 on the MEMPTR.

handle

An X-document object handle or X-noderef object handle. If the specified handle contains XML text, the AVM deletes the existing text first.

longchar

A LONGCHAR variable to contain the XML Schema document text in memory.

The AVM saves the XML Schema document text to the LONGCHAR variable in the code page that corresponds to the character encoding you specify in the *encoding* option. If you do not specify a character encoding for the XML Schema document text, the AVM saves the LONGCHAR variable in UTF-8.

If the LONGCHAR variable's code page is fixed (that is, set using the FIX-CODEPAGE statement) and the fixed code page is not equivalent to the character encoding you specify in the *encoding* option, the WRITE-XMLSCHEMA( ) method returns an error and the XML Schema document is not saved to the LONGCHAR.

formatted

An optional LOGICAL expression where TRUE directs the AVM to format the XML Schema document text in a hierarchical manner using extra white space, carriage returns, and line feeds. The default value is FALSE.

If you specify the Unknown value (?), the method uses the default value of FALSE.

encoding

An optional CHARACTER expression that specifies the name of the character encoding the AVM uses to write the XML Schema document text. The default encoding is UTF-8.

The encoding name must be an Internet Assigned Numbers Authority (IANA) name supported by the ABL XML Parser. For a list of supported IANA encodings and their corresponding ABL code pages, see Table 77: IANA encodings and corresponding ABL code pages in the ENCODING attribute reference entry.

Note: The AVM records this character encoding in the encoding declaration in the XML document's prologue. If you specify the empty string ("") or the Unknown value (?), the AVM uses the default encoding of UTF-8. In this case, the AVM does not record the character encoding in the XML document's encoding declaration.

If *target-type* is "HANDLE", the X-document's ENCODING attribute is also set.

min-xmlschema

An optional LOGICAL expression where TRUE directs the AVM to write the minimum amount of schema for the object, and FALSE directs the AVM to write the complete schema including ABL-specific schema attributes. The default value is FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

When TRUE, ABL-specific schema information (such as, field format, non-unique indexes, and so on) is omitted from the XML Schema. If the ABL data type of the temp-table field is not the default ABL data type for the XML Schema type, the AVM writes the `prodata:`*`dataType`* XML Schema attribute for the field. If the initial value of the temp-table field is TODAY, NOW, or UNKNOWN (and UNKNOWN is not the default initial value for the field's data type), the AVM writes the `prodata:`*`initial`* XML Schema attribute for the field.

When TRUE, the XML Schema will contain any ABL-specific XML Schema attributes needed to define the data relations for a ProDataSet.

omit-initial-values

An optional LOGICAL expression where TRUE directs the AVM to indicate in the XML Schema that the fields are optional in the XML instance document, and FALSE directs the AVM to indicate in the XML Schema that the fields are required in the XML instance document. The default value is FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

This behavior applies both to temp-table fields that have the default initial value for its data type and for fields that have an initial value set with the ABL INITIAL option.

Note: See the entry in WRITE-XML( ) method for this argument for considerations of using this argument with XML data.

If the NAMESPACE-URI attribute value for a temp-table within a ProDataSet object is different than that of the ProDataSet object, the method creates a separate XML Schema file for the temp-table definition. The namespace URI for the temp-table is imported into the ProDataSet schema, with a `schemaLocation` pointing to a separate XML Schema file containing the temp-table definition. Multiple namespaces are supported only when *target-type* is "FILE". If you specify multiple namespaces and *target-type* is not "FILE", the method generates an error and returns FALSE.

You can specify how a temp-table column is represented in XML Schema (that is, as an ELEMENT, ATTRIBUTE, or TEXT) by:

- Setting the XML-NODE-TYPE attribute on the Buffer-field object handle
- Specifying the XML-NODE-TYPE option on the DEFINE TEMP-TABLE statement

When writing schema for a ProDataSet object that contains data-relations, you can nest child rows of a ProDataSet buffer definition within their parent buffer definitions in the resulting XML Schema by:

- Setting the NESTED attribute on the data-relation object handle to TRUE
- Specifying the NESTED option for the data-relation on the DEFINE DATASET statement
- Specifying the NESTED option in the ADD-RELATION( ) method
- Specifying the FOREIGN-KEY-HIDDEN option to prevent duplicating the key fields in the child records that are nested within the parent record

If your temp-tables contain array fields, third party products utilizing the XML Schema might not map the ABL array field to an array column or object. For best interoperability with third party products, flatten array fields into individual fields.

You cannot write an XML representation of the schema for a database buffer.

The following code example defines a static ProDataSet object and writes the ProDataSet object schema to an XML Schema file:

```
DEFINE VARIABLE cTargetType AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFile       AS CHARACTER NO-UNDO.
DEFINE VARIABLE lFormatted  AS LOGICAL   NO-UNDO.
DEFINE VARIABLE cEncoding   AS CHARACTER NO-UNDO.
DEFINE VARIABLE lMinSchema  AS LOGICAL   NO-UNDO.
DEFINE VARIABLE lRetOK      AS LOGICAL   NO-UNDO.

DEFINE TEMP-TABLE ttCustomer NO-UNDO LIKE Customer.
DEFINE TEMP-TABLE ttOrder    NO-UNDO LIKE Order.
DEFINE TEMP-TABLE ttInvoice  NO-UNDO LIKE Invoice.

DEFINE DATASET DSET FOR ttCustomer, ttOrder, ttInvoice
  DATA-RELATION CustOrd FOR ttCustomer,
    ttOrd RELATION-FIELDS(CustNum, CustNum) NESTED
  DATA-RELATION OrdInv FOR ttOrder,
    ttInv RELATION-FIELDS(OrderNum, OrderNum) NESTED.

ASSIGN
  cTargetType = "file"
  cFile       = "cust-ord-inv.xsd"
  lFormatted  = TRUE
  cEncoding   = ?
  lMinSchema  = FALSE.

lRetOK = DATASET DSET:WRITE-XMLSCHEMA(cTargetType, cFile, lFormatted,
  cEncoding, lMinSchema).
```

The following code example defines a static temp-table object, and writes the temp-table object schema to an XML Schema file:

```
DEFINE VARIABLE cTargetType AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFile       AS CHARACTER NO-UNDO.
DEFINE VARIABLE lFormatted  AS LOGICAL   NO-UNDO.
DEFINE VARIABLE cEncoding   AS CHARACTER NO-UNDO.
DEFINE VARIABLE lMinSchema  AS LOGICAL   NO-UNDO.
DEFINE VARIABLE lRetOK      AS LOGICAL   NO-UNDO.

DEFINE TEMP-TABLE ttCust NO-UNDO LIKE Customer.

ASSIGN
  cTargetType = "file"
  cFile       = "ttCust.xsd"
  lFormatted  = TRUE
  cEncoding   = ?
  lMinSchema  = FALSE.

lRetOK = TEMP-TABLE ttCust:WRITE-XMLSCHEMA(cTargetType, cFile, lFormatted,
  cEncoding, lMinSchema).
```

See also

ENCODING attribute, FIX-CODEPAGE statement, FOREIGN-KEY-HIDDEN attribute, NAMESPACE-PREFIX attribute, NAMESPACE-URI attribute, NESTED attribute, READ-XML( ) method, READ-XMLSCHEMA( ) method, WRITE-XML( ) method, XML-NODE-TYPE attribute

X attribute

The pixel location of the left edge of a widget relative to the left edge of the parent widget or the display. The pixel location of the mouse cursor relative to the left edge of the display (for the last mouse event).

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LAST-EVENT system handle, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

This attribute is read-only for field groups, browse cells, and the LAST-EVENT handle.

For all user interface widgets except windows, the X attribute specifies the location, in pixels, of the left edge of the widget relative to the left edge of its parent widget. In windows, it is the location of the left edge of the window relative to the left edge of the display.

For a browse column, the X attribute returns the Unknown value (?) if the column is hidden.

For control-frames, the X attribute maps to the Left property of the control-frame COM object (ActiveX control container).

For the LAST-EVENT handle, the X attribute returns the pixel location of a mouse event relative to the left edge of the current frame.

This attribute is functionally equivalent to the COLUMN attribute.

X-DOCUMENT attribute

Contains the X-document object handle of an XML document posted to the transaction server or the Unknown value (?) if there isn't one.

Data type: HANDLE

Access: Read-only

Applies to: WEB-CONTEXT system handle

XCODE-SESSION-KEY attribute

Data type: CHARACTER

Access: Write-only

Applies to: SECURITY-POLICY system handle

When set, the XCODE-SESSION-KEY attribute contains the encryption key used by the COMPILE statement for the current session. The attribute must contain the same key used with the XCODE utility. The attribute does not have a default or initial value. To unset the attribute, set it with the Unknown value (?).

Because the attribute contains an encryption key, any attempt to read the attribute is a compiler error.

Both the XCODE utility and this attribute have an eight-character limit for keys. However, neither the utility or the attribute issues a warning or error if the key is longer. Instead, the utility and attribute use the first eight characters and ignore additional characters.

The XCODE utility does not perform code page conversions and does not use the -cpinternal parameter when encrypting files. Therefore, the source code and key will use the default code page of the operating system where you run the XCODE utility. If a different code page is in effect where XCODE-SESSION-KEY is set, then code page conversions may prevent the attribute key from matching the XCODE utility key and the compile fails. To prevent this case, use only US-ASCII characters, which are found in all code pages below code point 128.

You cannot use XCODE with the XREF, XREF-XML, STRING-XREF, or LISTING options together. Also, if the DEBUG-LIST option is used with an encrypted source file, the resulting debug file will only contain a notice that the source file is encrypted.

The LIST-QUERY-ATTRS and LIST-SET-ATTRS functions do not return the XCODE-SESSION-KEY attribute when given a SECURITY-POLICY handle.

XML-DATA-TYPE attribute

Returns the XML Schema data type for the temp-table buffer-field object.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer-field object handle

The XML Schema data type must be compatible with the ABL data type for the field.

If the temp-table schema was created from an XML Schema, this attribute is the same as the `xsd:type` attribute in the XML Schema.

This attribute does not apply to database buffer fields.

For more information about the ABL XML data type mapping rules, see *Use XML with ABL Applications*.

XML-ENTITY-EXPANSION-LIMIT attribute
(WebSpeed Only)

Indicates the maximum number of entity substitutions the XML parser (either DOM or SAX) will permit in a XML document sent as a WebSpeed request.

Data type: INTEGER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

By limiting the number of entity substitutions, use of this attribute prevents recursive entities from consuming system resources. The default value is 50,000. The attribute's value must be equal to or greater than 0; setting the attribute to a negative value will result in a runtime error. If the attribute is set to the Unknown value (?), the feature is off.

You can set this behavior globally by using the `-entityExpansionLimit` startup parameter. The default value of the attribute is set by the startup parameter, but you can change the attribute as needed to override the behavior set by the startup parameter. See the *Startup Command and Parameter Reference* for more information.

What XML-ENTITY-EXPANSION-LIMIT avoids

In WebSpeed, the first time you access the X-document handle or any of its attributes, you trigger a load of the document, which precedes your access. For example, if you set the ENTITY-EXPANSION-LIMIT attribute of X-document (of WEB-CONTEXT) before accessing X-document (of WEB-CONTEXT) or any of its attributes, when the document arrives, WebSpeed first loads the document, then sets ENTITY-EXPANSION-LIMIT to your value. So at load time, ENTITY-EXPANSION-LIMIT might not contain your value, which might cause WebSpeed not to observe the entity expansion limit when resolving entities.

By contrast, if you set XML-ENTITY-EXPANSION-LIMIT instead of ENTITY-EXPANSION-LIMIT, when the document arrives, WebSpeed assigns the value of XML-ENTITY-EXPANSION-LIMIT to ENTITY-EXPANSION-LIMIT before doing anything else. As a result, WebSpeed correctly observes the entity expansion limit.

How to use XML-ENTITY-EXPANSION-LIMIT

In WebSpeed, instead of accessing the ENTITY-EXPANSION-LIMIT attribute of an X-document (of WEB-CONTEXT), access XML-ENTITY-EXPANSION-LIMIT. For example, if you want a WebSpeed application to limit the number of entity expansions for the WEB-CONTEXT:X-DOCUMENT, have the application set the WEB-CONTEXT:XML- ENTITY-EXPANSION-LIMIT attribute before accessing the WEB-CONTEXT:X-DOCUMENT.

XML-NODE-NAME attribute

Indicates the name of the XML element or attribute representing the ProDataSet, the temp-table, the temp-table buffer, or the temp-table buffer-field object name in an XML document.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer object handle, Buffer-field object handle, ProDataSet object handle, Temp-table object handle

This attribute's purpose overlaps with the SERIALIZE-NAME attribute. Because of this overlap, the attributes interact as follows:

- The READ-XML( ) and WRITE-XML( ) methods always use the XML-NODE-NAME attribute value. If you set a value for the XML-NODE-NAME attribute, it keeps that value regardless of how you set the SERIALIZE-NAME attribute.
- If you do not set the XML-NODE-NAME attribute and set the SERIALIZE-NAME attribute, the AVM sets XML-NODE-NAME equal to SERIALIZE-NAME.
- If you do not set either attribute, the AVM sets both to the ABL object name.

This attribute allows you to work around ABL names that use illegal XML characters or to work around XML element or attribute names that are reserved words in ABL.

This attribute does not apply to database buffers and fields.

For more information about using this attribute, see *Use XML with ABL Applications*.

See also

READ-XML( ) method, SERIALIZE-NAME attribute, WRITE-XML( ) method

XML-NODE-TYPE attribute

The XML node type of the temp-table buffer-field or ProDataSet object, which lets you specify how the object is represented in XML and XML Schema.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer-field object handle, ProDataSet object handle

The following table lists the valid XML node types for a buffer-field object.

Table 114: XML node types for buffer-field objects

When the XML node type is . . .	The buffer field is . . .
`"ATTRIBUTE"`	Represented as an attribute of the temp-table element in both the XML Schema and data.
`"ELEMENT"`	Represented as a child element of the temp-table element in both the XML Schema and data.
`"HIDDEN"`	Omitted from both the XML Schema and data.
`"TEXT"`	Represented as a text element in both the XML Schema and data. **Note:** Each table can contain only one `"TEXT"` field. When a table contains a `"TEXT"` field, it cannot contain `"ELEMENT"` fields; it can contain only `"ATTRIBUTE"` fields. A table that contains a `"TEXT"` field cannot be part of a nested data-relation.

The XML node type of a buffer field that represents an array must be either "ELEMENT" or "HIDDEN".

The following table lists the valid XML node types for a ProDataSet object.

Table 115: XML node typesfor ProDataSet objects

When the XML node type is . . .	The ProDataSet name is . . .
"ELEMENT"	Represented as the root element in both the XML Schema and data.
"HIDDEN"	Omitted from both the XML Schema and data.

This attribute's purpose overlaps with the SERIALIZE-HIDDEN attribute. Because of this overlap, the attributes interact as follows:

- The WRITE-XML( ) method always uses the XML-NODE-TYPE attribute value. If you set a value for the XML-NODE-TYPE attribute, it keeps that value regardless of how you set the SERIALIZE-HIDDEN attribute.
- If you do not set the XML-NODE-TYPE attribute and set the SERIALIZE-HIDDEN attribute to TRUE, the AVM sets XML-NODE-TYPE to "HIDDEN".
- If you do not set the XML-NODE-TYPE attribute and set the SERIALIZE-HIDDEN attribute to FALSE, the AVM sets XML-NODE-TYPE to "ELEMENT".
- If you do not set either attribute, the AVM sets XML-NODE-TYPE to "ELEMENT" and sets SERIALIZE-HIDDEN to FALSE.

This attribute does not apply to database buffer fields.

See also

SERIALIZE-HIDDEN attribute, WRITE-XML( ) method, WRITE-XMLSCHEMA( ) method

XML-SCHEMA-PATH attribute
(WebSpeed Only)

A delimiter-separated list of directory paths for the XML Document Type Definition (DTD) associated with a particular XML document in a WebSpeed environment. Searched if the XML document contains a relative path to the DTD.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

Almost identical to the SCHEMA-PATH attribute of the X-document and SAX-reader objects. Like SCHEMA-PATH, the pathnames can contain Unicode characters. See *Internationalize ABL Applications* for more information about Unicode. For more information on SCHEMA-PATH, see the reference entry for the SCHEMA-PATH attribute.

What XML-SCHEMA-PATH avoids

In WebSpeed, the first time you access the X-document handle or any of its attributes, you trigger a load of the document, which precedes your access. For example, if you set the SCHEMA-PATH attribute of X-document (of WEB-CONTEXT) before accessing X-document (of WEB-CONTEXT) or any of its attributes, when the document arrives, WebSpeed first loads the document, then sets SCHEMA-PATH to your value. So at load time, SCHEMA-PATH might not contain your value, which might cause WebSpeed not to find your DTD, which might cause validation of the document to fail.

By contrast, if you set XML-SCHEMA-PATH instead of SCHEMA-PATH, when the document arrives, WebSpeed assigns the value of XML-SCHEMA-PATH to SCHEMA-PATH before doing anything else. As a result, WebSpeed searches for your DTD.

How to use XML-SCHEMA-PATH

In WebSpeed, instead of accessing the SCHEMA-PATH attribute of X-document (of WEB-CONTEXT), access XML-SCHEMA-PATH. For example, if you want a WebSpeed application to set the DTD search path (perhaps based on a Web request) of an XML document, instead of having the application set the SCHEMA-PATH attribute of X-document (of WEB-CONTEXT), have the application set the XML-SCHEMA-PATH attribute (of WEB-CONTEXT).

For more information on accessing XML documents using the Document Object Model (DOM) and SAX interfaces, see *Use XML with ABL Applications*.

XML-STRICT-ENTITY-RESOLUTION attribute
(WebSpeed Only)

Indicates whether or not the XML parser (DOM or SAX) will attempt to resolve an external entity if the entity is located outside of the directories in the XML-SCHEMA-PATH attribute of an XML document sent as a WebSpeed request.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

If XML-STRICT-ENTITY-RESOLUTION is TRUE, the parser will NOT attempt to resolve an external entity that is located outside of the directories in the XML-SCHEMA-PATH attribute of the WEB-CONTEXT system handle. If XML-STRICT-ENTITY-RESOLUTION is FALSE, the parser will attempt to resolve an external entity. Setting this attribute to TRUE can help prevent an external caller from gaining access to data located in directories not included in the XML-SCHEMA-PATH attribute.

You can set this behavior globally by using the `-strictEntityResolution` startup parameter. The default value of the attribute is set by the startup parameter, but you can change the attribute as needed to override the behavior set by the startup parameter. See the *Startup Command and Parameter Reference* for more information.

What XML-STRICT-ENTITY-RESOLUTION avoids

In WebSpeed, the first time you access the X-document handle or any of its attributes, you trigger a load of the document, which precedes your access. For example, if you set the STRICT-ENTITY-RESOLUTION attribute of X-document (of WEB-CONTEXT) before accessing X-document (of WEB-CONTEXT) or any of its attributes, when the document arrives, WebSpeed first loads the document, then sets STRICT-ENTITY-RESOLUTION to your value. So at load time, STRICT-ENTITY-RESOLUTION might not contain your value, which might cause WebSpeed not to observe the XML-SCHEMA-PATH when resolving entities.

By contrast, if you set XML-STRICT-ENTITY-RESOLUTION instead of STRICT-ENTITY-RESOLUTION, when the document arrives, WebSpeed assigns the value of XML-STRICT-ENTITY-RESOLUTION to STRICT-ENTITY-RESOLUTION before doing anything else. As a result, WebSpeed correctly observes the XML-SCHEMA-PATH.

How to use XML-STRICT-ENTITY-RESOLUTION

In WebSpeed, instead of accessing the STRICT-ENTITY-RESOLUTION attribute of X-document (of WEB-CONTEXT), access XML-STRICT-ENTITY-RESOLUTION. For example, if you want a WebSpeed application to be restricted to XML-SCHEMA-PATH when resolving entities, have the application set the WEB-CONTEXT:XML-STRICT-ENTITY-RESOLUTION attribute before accessing the WEB-CONTEXT:X-DOCUMENT.

XML-SUPPRESS-NAMESPACE-PROCESSING attribute
(WebSpeed Only)

Specifies whether to suppress namespace processing.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

Almost identical to the SUPPRESS-NAMESPACE-PROCESSING attribute of the X-document handle. For more information on SUPPRESS-NAMESPACE-PROCESSING, see the reference entry for the SUPPRESS-NAMESPACE-PROCESSING attribute.

What XML-SUPPRESS-NAMESPACE-PROCESSING avoids

In WebSpeed, the first time you access the X-document handle or any of its attributes, either directly or indirectly, you trigger a load of the document, which precedes your access. For example, if you set the SUPPRESS-NAMESPACE-PROCESSING attribute of X-document (of WEB-CONTEXT) before accessing X-document (of WEB-CONTEXT) or any of its attributes, when the document arrives, WebSpeed first loads the document, then sets SUPPRESS-NAMESPACE-PROCESSING to your value. So at load time, SUPPRESS-NAMESPACE-PROCESSING might not be set to your value, which might cause document validation to fail.

By contrast, if you set XML-SUPPRESS-NAMESPACE-PROCESSING instead of SUPPRESS-NAMESPACE-PROCESSING, when the document arrives, WebSpeed assigns the value of XML-SUPPRESS-NAMESPACE-PROCESSING to SUPPRESS-NAMESPACE-PROCESSING before doing anything else. Even if this triggers a load of the document, SUPPRESS-NAMESPACE-PROCESSING already contains your value, so WebSpeed is able to validate your document.

How to use XML-SUPPRESS-NAMESPACE-PROCESSING

In WebSpeed, instead of accessing SUPPRESS-NAMESPACE-PROCESSING of X-document (of WEB-CONTEXT), access XML-SUPPRESS-NAMESPACE-PROCESSING (of WEB-CONTEXT). For example, if you want a WebSpeed application to turn namespace processing off (perhaps in response to a Web request), instead of having the application assign FALSE to SUPPRESS-NAMESPACE-PROCESSING of X-document (of WEB-CONTEXT) have the application assign FALSE to XML-SUPPRESS-NAMESPACE-PROCESSING (of WEB-CONTEXT).

For more information on accessing XML documents using the SAX and X-document interfaces, see *Use XML with ABL Applications*.

Y attribute

The pixel location of the top edge of the widget relative to the top edge of the parent widget or the display. The pixel location of the mouse cursor relative to the top edge of the display (for the last mouse event). This attribute is functionally equivalent to the ROW attribute.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse and cell), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LAST-EVENT system handle, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

This attribute is read-only for field groups, browse cells, and the LAST-EVENT handle.

For all user interface widgets except windows, the Y attribute specifies the location, in pixels, of the top edge of the widget relative to the top edge of its parent widget. In windows, it is the location of the top edge of the window relative to the top edge of the display.

For a browse column, the Y attribute returns the Unknown value (?) if the column is hidden.

For control-frames, the Y attribute maps to the Top property of the control-frame COM object (ActiveX control container).

For the LAST-EVENT handle, the Y attribute returns the pixel location of a mouse event relative to the top edge of the current frame.

YEAR-OFFSET attribute

The current start date for the ABL two-digit year-range of 100 years. Use this attribute to display DATE, DATETIME, and DATETIME-TZ data when the format specifies a two-digit year.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SESSION system handle

Typical values are 1920 or 1950. This attribute provides the same functionality as the Year Offset (-yy) parameter. The default value is 1950.

7

Handle-based Object Events Reference

Events are run-time conditions that cause the ABL Virtual Machine (AVM) to respond by executing specified ABL code or by activating certain system behaviors. Events can occur directly as a result of ABL, user, or other external actions that affect the application. However events occur, the resulting executed code or activated behavior is referred to as event driven.

There are a number of factors that determine how ABL interprets events. The most important factor is the type of widget or handle receiving the event. Some widget types have default system actions in response to certain events. For example, the default system action for the A event on a fill-in widget is to insert the letter A into the fill-in at the current cursor location; however, there is no default system action for the A event on a button widget.

Different widget attribute settings determine how ABL interprets and prioritizes events. If you enable a widget for direct manipulation, direct manipulation events take priority over all other events. For example, if you write a trigger for a CHOOSE event and another for a SELECT event on a selectable widget, the AVM only executes the SELECT event trigger when you click on that widget.

This section describes the following topics for user-interface events only:

- Introduction to ABL events
- Event tables

You may consider an event to be supported for all interfaces, on all operating systems, and for SpeedScript unless otherwise indicated in the reference entry. User-interface events do not apply to SpeedScript programming.

For information on the following events, see the relevant documentation:

- **DDE-NOTIFY** — *OpenEdge Programming Interfaces*
- **PROCEDURE-COMPLETE** — *ABL Reference*

The following descriptions refer to both compile-time and run-time behavior, features that the language generally supports and determines at compile time and actions directed by using these features at run time. When describing compile-time features or actions, this section references ABL or the ABL compiler. When describing ABL or event-driven actions taken at run time, this section references the AVM.

Introduction to ABL events

This section covers the following topics:

- Event priority
- Apply events
- Triggers and low-level keyboard events

Event priority

The priority of events is an important concept. For any mouse or keyboard action on a widget, ABL generates a single event. Thus, certain events take priority over others that are generated by the same keyboard or mouse action for the same widget. Without direct manipulation, the priority (first to last) of keyboard events is key label, key function, and then high-level widget events such as CHOOSE. The priority of mouse events is three-button, portable, and then high-level widget events. Within three-button and portable mouse events, low-level mouse events (up, down) take priority over high-level mouse events (click, double-click). For more information on keyboard and mouse event priority, see the topics on handling user input in *OpenEdge Programming Interfaces*.

Apply events

You can apply any event to any widget using the APPLY statement. Depending on the event-widget pair, the APPLY statement may or may not perform the default system action. Regardless of whether there is a default system action associated with an event-widget pair, you can write a trigger for the pair. The APPLY statement executes a trigger associated with an event-widget pair. If the event-widget pair has a default system action, that action occurs before or after the trigger executes, depending on the event.

The APPLY statement also serves as a communications/dispatch mechanism between procedures in an application. You can define a trigger for an event-procedure pair. For example:

```
ON CLOSE OF THIS-PROCEDURE DO:
  APPLY "CLOSE" TO WINDOW-1.
END.
```

To define a trigger for a procedure, specify any ABL event in an ON statement for a procedure handle. This capability allows you to encapsulate functionality in a procedure. To access that functionality, simply use the APPLY statement to apply the appropriate event to the handle of the procedure. For more information, see the APPLY statement reference entry.

When working with browse widgets, you can apply events to the browse widget and to a browse cell in the currently focused row. For example:

```
ON CHOOSE OF button1 DO:
  APPLY "ENTRY" TO my-browse IN FRAME a.
  /* Code to focus a particular row in the browse. */
  APPLY "ENTRY" TO column3 IN BROWSE my-browse.
END.
```

Since a browse cell is the intersection of a column and row, referencing the column name references the intersection of that column and the currently focused row.

Note: The most flexible technique for encapsulating functionality in a procedure is to define and call internal procedures of a persistent procedure.

Triggers and low-level keyboard events

Some low-level keyboard events cannot have associated triggers and maintain their default behavior at the same time. In general, if the AVM gets an event from the user interface system (UIS) that has a trigger associated with it, the AVM handles the default behavior and tells the UIS to ignore the event. This allows the AVM to cancel the default behavior in response to a RETURN NO-APPLY invoked by the trigger.

However, there are some low-level keyboard events for which the AVM does not handle the default behavior. These include the cursor keys, especially. When the AVM gets one of these events with an associated trigger, it tells the UIS to ignore the event as usual, but because the AVM does not handle the default behavior for the event, the standard UIS behavior is lost, as well. Thus, a cursor key event (for example, CURSOR-UP) that has an associated trigger does not move the cursor.

Note that for many low-level events, such as mouse button and printable character events in fill-in fields and editors, the AVM **does** provide the default handling. Triggers on these events have no effect on the default event behavior unless they return NO-APPLY. The same is true of keyboard events that generate high-level functions, such as TAB and RETURN.

For those low-level, non-printable, keyboard events that are not handled by the AVM, do not associate triggers with them unless you do not want the default behavior of the event. For those low-level events that have no standard UIS behavior (such as, programmable function keys) triggers have no negative effects, and in fact, are very useful in defining a program action. In general, check any questionable low-level events in a test procedure both before and after associating triggers with them to see if any standard behavior is affected. An empty trigger block is sufficient to detect differences in behavior. For example:

```
ON event ANYWHERE DO: END.
```

Event tables

The tables in this section describe user interface events, the user actions that generate the events, and widgets that have default behavior for the events. The term **field-level widgets** refers to any widgets that can be part of a field group in a frame: fill-ins, sliders, selection lists, toggle boxes, radio sets, editors, rectangles, images, text, buttons, combo boxes, and browse widgets. Frames, dialog boxes, windows, menus (including menu bars and pop-up menus), sub-menus, and menu items can also receive events. Note that there is frequently a distinction made between a browse widget and a single cell in an updateable browse. For the most part, a browse cell behaves as a fill-in widget.

The event tables in this section describe the following kinds of events:

- Keyboard events
- Mouse events
- High-level widget events
- Direct manipulation events
- Developer events
- Socket events
- ProDataSet events

Keyboard events

ABL makes all keyboard actions available as events that you can specify by either key label or key function. You can write triggers for these keyboard events, and associate these triggers with any field-level widget that receives input focus. For a complete list of key label and key function names, and information on how to use them, see the topics on handling user input in *OpenEdge Programming Interfaces*.

Keyboard events have default effects depending on the widget that receives the event. For example, the "A" key label event displays an uppercase "A" in a fill-in or editor widget, but has no default effect when applied to a button. ABL organizes some key function events into several classes that have default effects on selected groups of widgets. ABL also provides special keyboard events to write default triggers on classes of keys. You can use these default events to write a trigger for all keys in a particular class for which you have not defined a key label or key function event trigger.

Main classes of key function events

ABL supports three main classes of key function events:

- **Universal key function events** — Apply to all user-interface widgets except menus, submenus, and menu items
- **Navigation key function events** — Apply to those field-level widgets that can receive focus
- **Field editing key function events** — Apply to fill-ins and browse cells

Universal key function events

The following table describes universal key function events.

Table 116: Universal key function events

Event	Affected widgets	AVM action
BELL	All except control container, menu, menu item, and submenu	Trigger dependent (typically used to execute the BELL statement).
END-ERROR	All except menu, menu item, and submenu	For the first input operation of the program, raise the ENDKEY condition. For subsequent input operations, raise the ERROR condition.
ENDKEY	All except menu, menu item, and submenu	Raise the ENDKEY condition.
ERROR	All except control container, menu, menu item, and submenu	Raise the ERROR condition.
GO	All except menu, menu item, and submenu	Submit the input values for this frame.
HELP	All except menu, menu item, and submenu	Invoke application help.

Navigation key function events

The following table describes navigation key function events.

Table 117: Navigation key function events

Event	Affected widgets	AVM action
BACK-TAB	Browse, browse cell, button, combo box, control container, editor, fill-in, radio set, selection list, slider, toggle box	Move focus to the previous widget in the tab order within the current frame family.
CURSOR-DOWN DOWN	Browse, browse cell, button, combo box, control container, editor, fill-in, radio set, selection list, slider, toggle box	In a Windows GUI, move focus vertically to the next cell in a browse column, or to the next row of a browse widget when no cell is enabled. In the bottom-most browse cell or row, do not change focus. All other widgets provide no default AVM action. In character mode, move focus vertically down in the widget. If the cursor is in the bottom-most position of a widget, move focus to the first character position of the next widget below the current widget in the frame family. In the bottom-most position of the bottom-most widget in the frame family, do not change focus.

Event	Affected widgets	AVM action
CURSOR-LEFT LEFT	Browse, browse cell, button, combo box, control container, editor, fill-in, radio set, selection list, slider, toggle box	In a Windows GUI, move focus to the previous character position in the current cell of a browse widget. In the first character position of the cell, do not change focus. All other widgets provide no default AVM action. In character mode, move focus to the previous character position to the left in the current widget. In the left-most character position of the widget, move focus to the first character position of the next widget to the left within the frame, or if the current widget is the left-most widget in the frame, do not change focus.
CURSOR-RIGHT RIGHT	Browse, browse cell, button, combo box, control container, editor, fill-in, radio set, selection list, slider, toggle box	In a Windows GUI, move focus to the next character position in the current cell of a browse widget. In the last character position of the cell, do not change focus. All other widgets provide no default AVM action. In character mode, move focus to the next character position to the right in the current widget. In the right-most character position of the widget, move focus to the first character position of the next widget to the right within the frame, or if the current widget is the right-most widget in the frame, do not change focus.
CURSOR-UP UP	Browse, browse cell, button, combo box, control container, editor, fill-in, radio set, selection list, slider, toggle box	In a Windows GUI, move focus vertically to the previous cell in a browse column, or to the previous row of a browse widget when no cell is enabled. In the top-most browse cell or row, do not change focus. All other widgets provide no default AVM action. In character mode, move focus vertically up in the widget. If the cursor is in the top-most position of a widget, move focus to the first character position of the next widget above the current widget in the frame family. In the top-most position of the top-most widget in the frame family, do not change focus.
NEXT-FRAME	Browse, browse cell, button, combo box, editor, fill-in, radio set, selection list, slider, toggle box	Move focus to the next frame parented by the active window.

Event	Affected widgets	AVM action
PREV-FRAME	Browse, browse cell, button, combo box, editor, fill-in, radio set, selection list, slider, toggle box	Move focus to previous frame parented by the active window.
TAB	Browse, browse cell, button, combo box, control container, editor, fill-in, radio set, selection list, slider, toggle box	Move focus to the next widget in the tab order within the current frame family.

Field editing key function events

The following table describes field editing key function events.

Table 118: Field editing key function events

Event	Affected widgets	AVM action
BACKSPACE	Fill-in, browse cell	Delete one character to the left.
CLEAR	Fill-in, browse cell	Clear the current field value (character interfaces only).
DELETE-CHARACTER	Fill-in, browse cell	Delete one character to the right.
RECALL	Fill-in, browse cell	Restore the field to its value when it was last enabled.
RETURN	Fill-in, browse cell	Default behavior is different for character and graphical interfaces and dependent on the DATA-ENTRY-RETURN attribute of the SESSION handle.

Default keyboard events

ABL provides two keyboard events that you can use to write default triggers. The following table describes these events.

Table 119: Default keyboard events

Event	Affected widgets	Meaning
ANY-KEY	Browse, browse cell, button, combo box, editor, fill-in, radio set, selection list, slider, toggle box	Executes for any keyboard event for which the user has not defined a specific trigger.
ANY-PRINTABLE	Browse, browse cell, button, combo box, editor, fill-in, radio set, selection list, slider, toggle box	Executes for any keyboard event that normally produces a printable character.

Mouse events

ABL supports two types of mouse events—portable and three-button events. You can use portable mouse events to associate triggers with logical actions of any mouse. You can use the three-button mouse events to associate triggers with specific physical actions of a three-button mouse.

The following tables reference portable mouse buttons for portable mouse events and physical mouse buttons for three-button mouse events. For more information on the mapping between portable and physical mouse buttons and how the AVM processes mouse events, see the topics on handling user input in *OpenEdge Programming Interfaces*.

Portable mouse events

The following table lists the mouse events that apply to all mice, no matter how the buttons are configured.

Table 120: Portable mouse events

Event	User action	Affected widgets	AVM action
`MOUSE-SELECT-DOWN`	Press the mouse SELECT button.	All	Trigger dependent
`MOUSE-SELECT-UP`	Release the pressed mouse SELECT button.	All	Trigger dependent
`MOUSE-SELECT-CLICK`	Press and release the mouse SELECT button.	All	Trigger dependent
`MOUSE-SELECT-DBLCLICK`	Press and release the mouse SELECT button twice.	All	Trigger dependent
`MOUSE-MENU-DOWN`	Press the mouse MENU button.	All	Trigger dependent
`MOUSE-MENU-UP`	Release the pressed mouse MENU button.	All	Trigger dependent
`MOUSE-MENU-CLICK`	Press and release the mouse MENU button.	All	Trigger dependent
`MOUSE-MENU-DBLCLICK`	Press and release the mouse MENU button twice.	All	Trigger dependent
`MOUSE-EXTEND-DOWN`	Press the mouse EXTEND button.	All	Trigger dependent
`MOUSE-EXTEND-UP`	Release the pressed mouse EXTEND button.	All	Trigger dependent
`MOUSE-EXTEND-CLICK`	Press and release the mouse EXTEND button.	All	Trigger dependent
`MOUSE-EXTEND-DBLCLICK`	Press and release the mouse EXTEND button twice.	All	Trigger dependent

Event	User action	Affected widgets	AVM action
MOUSE-MOVE-DOWN	Press the mouse MOVE button. **Note:** In Windows, a MOUSE-SELECT-DOWN trigger defined for the same widget takes priority over MOUSE-MOVE-DOWN.	All	Trigger dependent
MOUSE-MOVE-UP	Release the pressed mouse MOVE button. In Windows, a MOUSE-SELECT-UP trigger defined for the same widget takes priority over MOUSE-MOVE-UP.	All	Trigger dependent
MOUSE-MOVE-CLICK	Press and release the mouse MOVE button. **Note:** In Windows, a MOUSE-SELECT-CLICK trigger defined for the same widget takes priority over MOUSE-MOVE-CLICK.	All	Trigger dependent
MOUSE-MOVE-DBLCLICK	Press and release the mouse MOVE button twice. **Note:** In Windows, a MOUSE-SELECT-DBLCLICK trigger defined for the same widget takes priority over MOUSE-MOVE-DBLCLICK.	All	Trigger dependent

Three-button mouse events

The following table lists the mouse events associated with physical mouse buttons.

Table 121: Three-button mouse events

Event	User action	Affected widgets	AVM action
LEFT-MOUSE-DOWN	Press the left mouse button.	All	Trigger dependent

Event	User action	Affected widgets	AVM action
LEFT-MOUSE-UP	Release the pressed left mouse button.	All	Trigger dependent
LEFT-MOUSE-CLICK	Press and release the left mouse button.	All	Trigger dependent
LEFT-MOUSE-DBLCLICK	Press and release the left mouse button twice.	All	Trigger dependent
RIGHT-MOUSE-DOWN	Press the right mouse button.	All	Trigger dependent
RIGHT-MOUSE-UP	Release the pressed right mouse button.	All	Trigger dependent
RIGHT-MOUSE-CLICK	Press and release the right mouse button.	All	Trigger dependent
RIGHT-MOUSE-DBLCLICK	Press and release the right mouse button twice.	All	Trigger dependent
MIDDLE-MOUSE-DOWN	Press the middle mouse button.	All	Trigger dependent
MIDDLE-MOUSE-UP	Release the pressed middle mouse button.	All	Trigger dependent
MIDDLE-MOUSE-CLICK	Press and release the middle mouse button.	All	Trigger dependent
MIDDLE-MOUSE-DBLCLICK	Press and release the middle mouse button twice.	All	Trigger dependent

High-level widget events

The following table lists high-level widget events. These are events generated by mouse or keyboard actions that perform high-level operations on a widget, such as entering a fill-in, choosing a button, or displaying a menu. Unless noted in the AVM Action column, triggers on these events execute before the AVM applies the event. If the trigger returns NO-APPLY, the AVM does not apply the event. If the trigger executes after the event takes place, NO-APPLY has no effect.

Note: If a CHOOSE, DEFAULT-ACTION, or VALUE-CHANGED event executes a trigger as a result of a mouse click that changes input focus, NO-APPLY will return focus to the widget that had focus prior to the event.

Table 122: High-level widget events

Event	User action	Affected widgets	AVM action
CHOOSE	A keyboard or mouse action that chooses a widget.	Button, non-toggle-box menu item	Trigger executes after choose takes place
DEFAULT-ACTION	A native keyboard or mouse event that confirms the selection of a value in a selection list or browse. In Windows applications, double-click a list item. In character applications, press **RETURN** or **DELETE-LINE**.	Selection list, Browse	Trigger dependent
DROP-FILE-NOTIFY[82], [83]	A mouse action that completes a drag-and-drop operation on a widget.	Browse, Button, Combo-box, Dialog-box, Editor, Fill-in, Frame, Radio-set, Selection-list, Slider, Toggle, Window	Trigger executes after drag-and-drop operation concludes **Note:** The trigger should call the END-FILE-DROP() method when it has finished processing all the files.
END	Occurs when the user repositions the browse to the end of the query's result set by pressing the **END** key.	Browse	Trigger dependent
END-SEARCH[82]	Occurs when an updateable browse ends a user-initiated search when a user either selects a row marker or clicks in a cell.	Browse	Trigger dependent

[82] Windows only.
[83] Graphical interfaces only.

Event	User action	Affected widgets	AVM action
ENTRY	A keyboard or mouse action that gives focus to the widget.	Browse, browse cell, button, combo box, control container, dialog box, editor, fill-in, frame, radio set, selection list, slider, toggle box, window	Trigger dependent **Note:** For a browse widget, ON ENTRY OF *browse-name* specifies a trigger for the browse widget and ON ENTRY OF *column-name* IN BROWSE *browse-name* specifies a trigger for a browse cell. The browse cell is the intersection of the named column and the currently focused row.
HOME	Occurs when the user repositions the browse to the beginning of the query's result set by pressing the **HOME** key.	Browse	Trigger dependent
ITERATION-CHANGED	A keyboard or mouse action that changes the current iteration of a browse. This event is obsolete; see the VALUE-CHANGED Event reference entry.	Browse	Trigger dependent
LEAVE	A keyboard or mouse action that takes focus from the widget.	Browse, browse cell, button, combo box, control container, dialog box, editor, fill-in, frame, radio set, selection list, slider, toggle box, window	Trigger dependent **Note:** For a browse widget, ON LEAVE OF *browse-name* specifies a trigger for the browse widget and ON LEAVE OF *column-name* IN BROWSE *browse-name* specifies a trigger for a browse cell. The browse cell is the intersection of the named column and the currently focused row.
MENU-DROP	A keyboard or mouse action that displays a menu.	Menu[84], submenu	Trigger dependent

[84] Supported only if the Menu POPUP-ONLY attribute is set to TRUE and the menu is set as a popup for some other widget.

Event	User action	Affected widgets	AVM action
OFF-END[85]	A keyboard or mouse action that tries to move after the last row of a browse.	Browse	Trigger dependent
OFF-HOME	A keyboard or mouse action that tries to move before the first row of a browse.	Browse	Trigger dependent
PARENT-WINDOW-CLOSE	An event that each descendant window receives when the common ancestor window in that family receives a WINDOW-CLOSE event.	Window	Trigger dependent
ROW-DISPLAY	Any browse action that results in a row being displayed in the browse.	Browse	Trigger dependent **Note:** The use of triggers for this event is restricted to special cases. When a row is displayed, use a trigger to modify attributes of individual cells in the column. It should be restricted to the following uses: changing cell colors, changing the cell font, referencing the cell in an expression, and (in Windows) changing the cell format.
ROW-ENTRY	A keyboard or mouse action that gives an updateable cell focus in a browse row.	Browse	Trigger dependent
ROW-LEAVE	A keyboard or mouse action that takes focus from the browse row where an updateable cell has focus.	Browse	Trigger dependent
SCROLL-HORIZONTAL[83]	A keyboard or mouse action that causes a horizontal scroll in a browse.	Browse	Trigger dependent **Note:** This event fires when a mouse or keyboard action scrolls a browse horizontally.[86]

[85] The OFF-END event can also occur when there are more rows to retrieve in the query on a ProDataSet temp-table buffer. For more information, see the ProDataSet events.

Event	User action	Affected widgets	AVM action
SCROLL-NOTIFY[83]	A mouse action in the scrollbar area of a browse.	Browse	Trigger dependent **Note:** This event allows the developer to track physical movement of the focused row in the browse viewport. This event is supported only for backward compatibility. Use SCROLL-HORIZONTAL and SCROLL-VERTICAL instead.
SCROLL-VERTICAL[83]	A keyboard or mouse action that causes a vertical scroll in a browse.	Browse	Trigger dependent **Note:** This event fires when a mouse or keyboard action scrolls a browse vertically.[86]
START-SEARCH[82]	A keyboard or mouse action that places an updateable browse into search mode.	Browse	Trigger dependent
VALUE-CHANGED	A keyboard or mouse action that changes the value of a widget. For the browse, any action that selects a row.	Browse, combo-box, editor (Windows GUI only), fill-in, radio set, selection list, slider, toggle box, toggle box menu item	Trigger executes after value changes
WINDOW-CLOSE	A keyboard or mouse action that causes the native window manager to close the affected window or dialog box.	Dialog box, window	Trigger dependent

[86] Only one SCROLL-VERTICAL or SCROLL-HORIZONTAL event will be generated each time the user scrolls the browse using the scroll bar, regardless of how many rows or columns the user scrolls through. However, each click of the mouse wheel counts as an independent scrolling event, so multiple events will be generated if the user rotates the wheel through multiple clicks. When scrolling is caused by keyboard actions, a SCROLL-VERTICAL or SCROLL-HORIZONTAL event will be generated after each key press, as long as the key press actually causes the browse to scroll. Holding a key down is equivalent to multiple key presses and will generate multiple events.

Event	User action	Affected widgets	AVM action
WINDOW-MAXIMIZED[82]	A keyboard or mouse action that causes the native window system to resize the window to its maximum size.	Window	Trigger executes after event takes place. However since the native system has control, a NO-APPLY does not stop the event from occurring. **Note:** This event occurs only in Windows.
WINDOW-MINIMIZED	A keyboard or mouse action that causes the native window system to minimize (iconify) a window and hide all of its descendant windows.	Window	Trigger executes after event takes place. However, since the native system has control, a NO-APPLY does not stop the event from occurring.
WINDOW-RESIZED	A keyboard or mouse action that causes the native window system to resize the window to any extent vertically or horizontally.	Window	Trigger executes after event takes place. However, since the native system has control, a NO-APPLY does not stop the event from occurring.
WINDOW-RESTORED	A keyboard or mouse action that causes the native window system to restore a window and any descendant windows to the state they were in before a prior maximize or minimize event.	Window	Trigger executes after event takes place. However since the native system has control, a NO-APPLY does not stop the event from occurring.

Direct manipulation events

Direct manipulation events are ABL events that directly modify the size, shape, position, and appearance of a widget. These events are generated by mouse actions. Each user interface widget either has direct manipulation enabled or does not. Some types of widgets, such as menus, cannot have direct manipulation enabled. You can enable widgets for direct manipulation by setting the SELECTABLE, MOVABLE, or RESIZABLE attribute to TRUE.

If a widget has direct manipulation enabled, then direct manipulation events take priority over all other events. In other words, while data manipulation is enabled, the widget cannot perform data entry or application control functions. For example, if you set SELECTABLE to TRUE for a button, ABL interprets a MOUSE-SELECT-UP event as a SELECTION event. If you set SELECTABLE to FALSE, ABL interprets the same event as a CHOOSE event.

Direct manipulation events can be broken down into two types: general and frame-only. General direct manipulation events apply to both field-level and frame widgets. Frame-only direct manipulation events apply only to frames.

The following sections list the ABL events associated with direct widget manipulation. The user actions listed for these events assume that you set the appropriate attributes to make each event possible. For example, a widget must be SELECTABLE to receive the SELECTION event.

General direct manipulation events

The following table lists the direct manipulation events that apply to field-level widgets and frames.

Table 123: General direct manipulation events

Event	User action	Affected widgets	AVM action
`DESELECTION`	For all selected widgets in a frame, click the mouse SELECT button on an unselected widget or in empty space in the frame. For a single selected widget, click the mouse EXTEND button on a selected widget.	Frame and field-level widgets with SELECTABLE attribute set to TRUE; browses.	**Internal:** Sets the widget's SELECTED attribute to FALSE. This setting takes effect after any trigger for the event executes. **Screen:** Removes the highlight from the affected widget or widgets.
`END-MOVE`	Release the pressed mouse MOVE button after moving the drag box for the widget or widgets.	Frame and field-level widgets with MOVABLE attribute set to TRUE; Also browse-columns.	**Internal:** Generates an END-MOVE event for each moved widget. **Screen:** Moves each widget to the new x and y coordinates of its drag box.
`END-RESIZE`	Release the pressed mouse SELECT button after stretching the resize box to resize the widget.	Frame and field-level widgets with RESIZABLE and SELECTABLE attributes set to TRUE; Also browse-columns.	**Internal:** Generates an END-RESIZE event for the resized widget. **Screen:** Resizes the widget to the new x and y coordinates of its resize box.
`END-ROW-RESIZE`	Release the pressed mouse SELECT button after resizing a row.	Browses.	**Internal:** Generates an END-ROW-RESIZE event for the resized row. **Screen:** Resizes all rows to the new height.
`SELECTION`	For a single unselected widget, click the mouse SELECT or EXTEND button on the widget. For multiple unselected widgets, release the pressed EXTEND button after drawing a select box around the widgets.	Frame and field-level widgets with SELECTABLE attribute set to TRUE.	**Internal:** Sets each widget's SELECTED attribute to TRUE. This setting takes effect after any trigger for the event executes. **Screen:** Highlights the affected widget or widgets.

Event	User action	Affected widgets	AVM action
START-MOVE	For a single widget, with the mouse pointer on the widget, press and hold the mouse MOVE button, and begin moving the mouse pointer. For multiple selected widgets—With the mouse pointer on any one of the selected widgets, press and hold the mouse MOVE button, and begin moving the mouse pointer.	Frame and field-level widgets with MOVABLE attribute set to TRUE; for multiple widgets, SELECTABLE attribute also set to TRUE; Also browse-columns.	**Internal:** Sends a START-MOVE event to all selected widgets. If the trigger returns a NO-APPLY, the AVM does not generate the subsequent END-MOVE event. **Screen:** Draws a drag box around each of the one or more affected widgets, and moves each drag box in the direction of the moving mouse pointer.
START-RESIZE	With the mouse pointer on a resize handle of a selected widget, press and hold the mouse SELECT button and begin moving the mouse pointer.	Frame and field-level widgets with RESIZABLE and SELECTABLE attributes set to TRUE; Browse columns.	**Internal:** Sends a START-RESIZE event to the selected widget. If the trigger returns NO-APPLY, the AVM does not generate the subsequent END-RESIZE event. **Screen:** Stretches a resize box around the widget in the direction of the moving mouse pointer.
START-ROW-RESIZE	With the mouse pointer on a row, press and hold the mouse SELECT button and begin moving the mouse pointer.	Browses.	**Internal:** Sends a START-ROW-RESIZE event to the browse. If the trigger returns NO-APPLY, the AVM does not generate the subsequent END-ROW-RESIZE event. **Screen:** Stretches a resize box around the row in the direction of the moving mouse pointer.

Frame-only direct manipulation events

The following table lists the direct manipulation events that apply only to frames.

Table 124: Frame-only direct manipulation events

Event	User action	Affected widgets	AVM action
EMPTY-SELECTION	Click the mouse SELECT button on an empty space in the frame.	Frame and dialog box, whether its SELECTABLE attribute is set to TRUE or FALSE.	**Internal:** Sends a DESELECTION event to all selected widgets in the frame and sends the EMPTY-SELECTION event to the frame. **Screen:** Removes the highlight around any selected widgets in the frame.
END-BOX-SELECTION	Release the pressed mouse SELECT or EXTEND button after moving the mouse pointer to stretch the select box.	Frame and dialog box with BOX-SELECTABLE attribute set to TRUE.	**Internal:** If the user pressed the mouse SELECT button, the AVM sends a SELECTION event to all widgets surrounded by the select box. If the user pressed a mouse EXTEND button, the AVM sends a SELECTION event to all unselected widgets, and a DESELECTION event to all selected widgets surrounded by the select box. If a trigger on END-BOX-SELECTION returns NO-APPLY, the AVM does not send a subsequent SELECTION or DESELECTION event. Note that this behavior differs from the behavior of END-MOVE and END-RESIZE. **Screen:** Erases the select box, highlights selected widgets, and removes the highlight from deselected widgets.
START-BOX-SELECTION	Press and hold the mouse SELECT or EXTEND button in an empty area of the frame and begin moving the mouse pointer.	Frame and dialog box with BOX-SELECTABLE attribute set to TRUE.	**Internal:** Sends a START-BOX-SELECTION event to the frame. If a trigger returns NO-APPLY, the AVM does not generate the subsequent END-BOX-SELECTION event. **Screen:** Draws a select box, which initially appears as a dot.

Developer events

ABL provides eleven events, labeled U1 through U10 and CLOSE, that you can invoke on any widget or handle using the APPLY statement. The only function of a developer event is the one provided by your own trigger definition.

Socket events

ABL looks for events to execute in the context of input-blocking or event-processing statements. During this processing if the AVM detects that data is available on a socket or that the remote end closed its socket or it detects that a client has connected to a port that the server has enabled connections to, a socket event is generated.

There are only two socket events, the READ-RESPONSE event, which applies only to socket objects, and the CONNECT event which applies only to server socket objects.

READ-RESPONSE event

AVM Detects — Data is available on a socket or the remote end of a connection has closed its socket; applies only to socket objects.

AVM Action — The AVM invokes the READ-RESPONSE event method or procedure.

The SET-CALLBACK() method is used to name the READ-RESPONSE event method or procedure and to associate it with a socket object. The AVM invokes this method or procedure whenever it detects that data is available on the socket or that the remote end of the socket has closed its end of the socket. In this method or procedure, the SELF handle identifies the affected socket object.

Note: The SET-READ-RESPONSE-PROCEDURE () can also be used to name the event procedure, but this is only for procedures. The SET-CALLBACK() method can be used for both procedures and methods, and is preferable.

To determine if the event method or procedure was invoked because data is available for reading or because of a disconnect, the application can use one of several methods:

- The CONNECTED( ) method returns FALSE if the socket is not connected to a port, TRUE if it is connected.
- The GET-BYTES-AVAILABLE( ) method returns zero if the socket is not connected to a port or the number of bytes available for reading if it is connected.
- The READ( ) method returns FALSE if the socket is not connected to a port. It returns TRUE and the read data if it is connected.

CONNECT event

AVM Detects — A client has connected to a port that the server has enabled connections to; applies only to server socket objects.

AVM Action — The AVM invokes the CONNECT event procedure.

The SET-CALLBACK( ) method is used to name the CONNECT event procedure or method and to associate it with a server socket object. The CONNECT event procedure or method must accept one input parameter of type HANDLE. This is the handle to the implicitly created socket object for this connection. It is via this socket object that the server communicates with the client.

Note: The SET-CONNECT-PROCEDURE( ) method can also be used to name the event procedure, but this is only for procedures. Progress recommends the SET-CALLBACK() method that can be used for both procedures and methods.

If the SET-CALLBACK( ) method is not invoked, or if it fails, no CONNECT event procedure or method will be executed when the CONNECT event occurs.

ProDataSet events

ABL provides events you can invoke to execute application-specific code that handles FILL operations on a ProDataSet object or Buffer object, as well as row-level change operations. You can use the SET-CALLBACK-PROCEDURE( ) method to associate an action with these events.

Event procedures must define a single parameter for the ProDataSet object (DATASET or DATASET-HANDLE) as an INPUT parameter BY-REFERENCE. This allows the event procedure to operate on the ProDataSet object using static ABL to reference its buffers and fields, without the ProDataSet object being physically copied. This also means that because the ProDataSet object is not copied, changes made to the ProDataSet object by the event procedure are made to the same copy used by all procedures.

The following sections describe the ProDataSet events.

- FILL events
- Row-level events
- OFF-END event
- FIND-FAILED event
- SYNCHRONIZE event

FILL events

There are two levels of FILL events: the first level is for a ProDataSet object or one of its member buffer objects; the second level is for individual records created in each temp-table.

The following table lists the first-level FILL events.

Table 125: First-level FILL events

Event	Affected objects	AVM action
AFTER-FILL	Buffer object of a DATASET temp-table, ProDataSet object	This event occurs at the very end of a FILL, and can be used to adjust the contents of the ProDataSet object or Buffer object, reject the FILL operation, or disconnect from a server or database. For a child table, the event occurs once for each parent record that is created.
BEFORE-FILL	Buffer object of a DATASET temp-table, ProDataSet object	This event occurs at the very beginning of a FILL, before anything is read or created. For a Buffer object, this event allows the developer to do preparatory work for an individual table. For the parent table in a set of related tables, where the FILL event is applied to this top-level table, it could be the same kind of connection code as for the ProDataSet object as a whole. For a child table, the event occurs once for each parent record that is created, and allows the developer to adjust the query for the child table, or cancel the FILL for that parent altogether. For a ProDataSet object, this event allows the developer to make a server or database connection, or do other preparatory work. Alternatively, it allows the developer to intercept and fully replace the default behavior.

The following table lists the second-level FILL events. These events occur once immediately before or after each record is created in a temp-table during a FILL.

Table 126: Second-level FILL events

Event	Affected objects	AVM action
AFTER-ROW-FILL	Buffer object of a DATASET temp-table	This event occurs after a record is created in the temp-table. The procedure can, for example, modify field values in the record by supplying values for calculated fields, or perform filtering and reject a record by deleting it. The procedure cannot modify record currency using the ProDataSet object buffers in any other way. It can use separately defined buffers to modify the ProDataSet object in other ways. The procedure can RETURN ERROR to abort the entire FILL, or RETURN NO-APPLY to cancel the cascading of the FILL to child buffers, if any.
BEFORE-ROW-FILL	Buffer object of a DATASET temp-table	This event occurs before a record is created in the temp-table, but after the data source record(s) for it are read. For example, this procedure could examine the database buffers or other information and decide not to create the record, using a RETURN NO-APPLY statement.

Row-level events

Row-level events are defined for making local changes to the records in a ProDataSet member buffer object. The following table lists the row-level events.

Table 127: Row-level events

Event	Affected objects	AVM action
ROW-CREATE	Buffer object of a DATASET temp-table	This event occurs immediately after the record is created in the temp-table. The current buffer for the temp-table is available and contains initial values as defined in the temp-table definition (or inherited from the schema). You can use this event to calculate initial values for fields, make changes to other records, or reject the creation by deleting the new temp-table record.

Event	Affected objects	AVM action
ROW-DELETE	Buffer object of a DATASET temp-table	This event occurs when you delete a temp-table record, immediately before the record is deleted. The event procedure can use this event to RETURN ERROR to cancel the delete, or to make adjustments to other records based on the delete. Since the record has not yet been deleted, the record is in the temp-table buffer and the code can look at its values. Because the code can assume that the DELETE will go through unless cancelled by the event procedure itself, it can take actions based on the record deletion while the record is still there to be looked at.
ROW-UPDATE	Buffer object of a DATASET temp-table	This event occurs immediately before the record is updated in the temp-table. It typically occurs when: • The buffer scope ends • The transaction scope ends • The RELEASE statement or BUFFER-RELEASE( ) method is run on the buffer • The buffer is needed for another record The AVM sets the SELF system handle to the handle of the buffer on which the event procedure is running before calling the event handler. If the event handler returns NO-APPLY or ERROR, the return is ignored. The handler has run, and it is too late to undo any changes to the record. You can use this event to determine if and how a record has changed by reading the buffer in the before-image table (using the SELF:BEFORE-ROWID attribute) and comparing it to the updated buffer. You can also use this event in the event handler to update fields in the record (for example, to supply a calculated field). You cannot read another record into the buffer on which the event procedure is running in the event handler. If you need to read another record, use a different buffer to avoid disturbing the record you are currently updating.

OFF-END event

The OFF-END event occurs when you position a query on a ProDataSet temp-table buffer past the last row. You can use this event to retrieve additional data source rows to add at the bottom of a ProDataSet temp-table (for example, in batches when there are too many data source rows to retrieve at one time).

The OFF-END event can also occur when the user performs a keyboard or mouse action in a browse that scrolls off the end (past the last row) of a browse on a ProDataSet temp-table buffer. For more information about using the OFF-END event with a browse, see High-level widget events.

Note: The OFF-END event is similar to the QUERY-OFF-END attribute, which is set to TRUE whenever the associated query object is positioned past the last row. The difference is that you must test the QUERY-OFF-END attribute for this condition at a specific place in your application code, whereas the OFF-END event procedure executes like a trigger whenever the event occurs.

Consider the following restrictions when using the OFF-END event with a query on a ProDataSet temp-table buffer:

- You can attach these events only to a query on a single ProDataSet temp-table buffer. You cannot attach these events to a query on a database buffer, or a query that involves a join.
- The query must be a scrolling query.
- If you never RETURN NO-APPLY, from the OFF-END event handler, the query will infinitely loop.
- Call the SET-CALLBACK-PROCEDURE( ) method before the query is opened.
- If you use the GET LAST statement or GET-LAST( ) method to get the last record associated with the query, the event handler is called repeatedly until it **does not** RETURN NO-APPLY (indicating that all records have been retrieved). For this reason, use caution when offering users the GET LAST action.
- The INDEXED-REPOSITION option is ignored for the query.

FIND-FAILED event

The FIND-FAILED event occurs when a FIND on a ProDataSet temp-table buffer fails. This can be the result of the FIND statement, the FIND-FIRST() method, or the FIND-UNIQUE() method. FIND-FAILED is NOT fired for FIND NEXT, FIND PREV, or FIND LAST statements, the CAN-FIND function, or the FIND-LAST( ) method.

You can use this event to adjust the contents of the ProDataSet object. The event handler must be able to determine the action to take based on the context of the ProDataSet object, and must RETURN NO-APPLY to indicate the action was successful. For example, when the event occurs, the event handler could retrieve or create a missing row or a set of related rows into the temp-table and RETURN NO-APPLY so the AVM can redo the operation that raised the event and continue processing.

SYNCHRONIZE event

The SYNCHRONIZE event occurs when a ProDataSet temp-table buffer is synchronized. That is, when the SYNCHRONIZE( ) method is run on the buffer or a parent buffer, or the buffer is selected in a browse. The event handler is invoked recursively at every level of the ProDataSet object hierarchy just before the recursion to the child levels.

By default, if the query is associated with a browse, the synchronize action of reopening the query automatically refreshes the browse. If the query is not associated with a browse, the synchronize action automatically gets the first row in the query by invoking a GET FIRST operation. If there is a REPOSITION data relation and no browse, the synchronize action gets the next record in the query by invoking a GET NEXT operation. Once these actions attempt to populate the buffer at a particular level, the SYNCHRONIZE event runs before moving recursively to the next lower level.

This event allows you to fetch rows, display buffer values in a frame, or take some other action. The handler procedure can also RETURN NO-APPLY to cancel the cascading of the synchronization to child buffers.

8

Class, Interface, and Enumeration Reference

This section contains reference entries that describe the built-in classes, interfaces, and enumerations supported in ABL.

Information about the object-oriented ABL interfaces, classes, and enumerations can be found in the ABL API doc at https://documentation.progress.com/output/oehttpclient/oe124/index.html. Other references and sources for the ABL API are summarized in the table below:

Procedure library and primary packages	Description	More information
dataadmin.pl • OpenEdge.DataAdmin	Contains the database administration API for OpenEdge Management (OEM). Contains security token service (STS) utilities.	See "Database Administration Entity Interface Reference" in *OpenEdge Programming Interfaces*.
OpenEdge.BusinessLogic.pl • Ccs.BusinessLogic • OpenEdge.BusinessLogic	Contains basic business entity classes and supporting code. Contains Common Component Specification (CCS) business entity interfaces and related types.	For Ccs.BusinessLogic see https://github.com/progress/CCS/tree/master/Specs/BusinessEntity/v1_0.
OpenEdge.BusinessRules.pl • OpenEdge.BusinessRules	Contains an ABL interface for calling Corticon Decision Services.	
OpenEdge.Core.pl • Ccs.common • Ccs.ServiceManager • OpenEdge.Core	Contains the most commonly used components including logging, and service, session, and startup manager interfaces.	For Ccs.common and Ccs.ServiceManager see https://github.com/progress/CCS/tree/master/Specs/Startup_Session_Service_Managers/v1_0.

Procedure library and primary packages	Description	More information
• OpenEdge.Logging • OpenEdge.Security		
OpenEdge.Net.pl • OpenEdge.Mobile • OpenEdge.Net • OpenEdge.Web	Contains common utilities and data writers for WebHandlers and HTTP clients.	
OpenEdge.ServerAdmin.pl • OpenEdge.ApplicationServer • OpenEdge.Rest	Contains administrative classes for querying and working with application server agents.	
OpenEdge.Security.pl • OpenEdge.Security	Contains classes and other types for working with the OpenEdge Authentication Gateway (OEAG). Includes interfaces for the policy and event callbacks.	See "Configuring event callbacks" in *Learn about the OpenEdge Authentication Gateway*.

The reference entries in this section include both native ABL classes and interfaces and OpenEdge .NET classes and interfaces. For information on working with ABL classes and interfaces, see *Develop Object-oriented ABL Applications*. For information on working with ABL enumerations, see the ENUM statement reference entry. For information on working with .NET classes and interfaces, see *Use GUI for .NET Objects in ABL Applications*.

The built-in ABL classes and interfaces support access to user-defined ABL classes and interfaces and also provide the foundation for structured error handling in ABL. They also support features of the ABL model for accessing .NET objects, including the handling of .NET exceptions using ABL structured error handling. For information on the properties and methods of these built-in ABL classes and interfaces, see the Class Properties and Methods Reference.

The built-in ABL enumerations support ABL's reflection capabilities. For example, the Progress.Reflect.AccessMode enumeration has Public, Protected, and Private members that can be used as parameters to specify the access mode of the class member(s) for which information is being requested. The entry for each built-in enumeration describes the members defined by that enumeration.

The built-in OpenEdge .NET classes and interfaces support ABL access to .NET objects. Some of the OpenEdge .NET classes extend Microsoft .NET classes. The reference entries for these classes list the OpenEdge built-in and extended properties, methods, and events that allow you to monitor and control the behavior of these class-based objects. For more information on the properties and methods listed for each OpenEdge .NET class and interface, see the Class Properties and Methods Reference. For more information on the .NET events listed for each OpenEdge .NET class, see the Class Events Reference in the *ABL Reference*. For information on the Microsoft .NET properties, methods, and events not listed in this manual for an extended Microsoft .NET class, see the .NET Framework class library.

ABL views all .NET classes and interfaces as ABL object types by treating the .NET root class, System.Object, as an immediate subclass of the ABL root class, Progress.Lang.Object. In this way, ABL merges the .NET type system with the ABL object type system, making the .NET type system act from within ABL as an extension of Progress.Lang.Object. However, .NET does not have any knowledge of the ABL type system. That is, ABL extends its own type system with .NET, but .NET does not recognize object types defined in ABL. For more information on ABL support for .NET types, see the Data types reference entry.

In addition to the OpenEdge .NET classes specifically referenced in this book, OpenEdge provides a set of third party .NET controls (OpenEdge Advanced UI Controls) for use with .NET forms and control containers. These Advanced UI Controls are supported for access as visual design objects in the Visual Designer of Progress Developer Studio for OpenEdge. For more information, see the appendix that lists these controls in *Use GUI for .NET Objects in ABL Applications*.

The following descriptions refer to both compile-time and run-time behavior, features that the language generally supports and determines at compile time and actions directed by using these features at run time. When describing ABL compile-time features or actions, ABL or the ABL compiler is the actor. When describing ABL or .NET actions taken at run time, the AVM (ABL Virtual Machine) or .NET (respectively) is the actor.

Progress.ApplicationServer.AdapterTypes enumeration

This enumeration describes the transport used for the current request executing on a Progress Application Server.

This class is FINAL and cannot be inherited.

Serializable:

Yes

Super Class

Progress.Lang.Enum class

Members

Member	Description
`Unexpected`	Invalid or unexpected transport
`APSV`	OpenEdge application server transport
`SOAP`	SOAP transport
`REST`	REST transport
`WEB`	HTTP transport

Public Methods

GetEnum( ) method	–

See also

AdapterType property

Progress.Data.BindingSource class
(Windows only; GUI for .NET only)

A BindingSource object (the ProBindingSource) that allows you to bind a .NET control to one of the following ABL data source objects:

- A query involving a temp-table, dataset, or database table (which can be a join query)
- A Buffer object

Note: Progress Software Corporation recommends that you open (or reopen) any query associated with a ProBindingSource instance using the PRESELECT option, which optimizes query performance.

The Progress.Data.BindingSource class is an OpenEdge .NET class that extends the .NET System.Windows.Forms.BindingSource class. The ProBindingSource acts as a conduit between a .NET control and an ABL data source object. A bound .NET control considers the ProBindingSource to be its data source. In turn, the ProBindingSource sets the underlaying ABL data source object as its data source. For a complete overview of data binding between .NET controls and ABL data source objects, see *Use GUI for .NET Objects in ABL Applications*.

Serializable:

No

Constructors

```
PUBLIC BindingSource ( INPUT query-hdl AS HANDLE )

PUBLIC BindingSource ( INPUT query-hdl AS HANDLE,
                       INPUT include-fields AS CHARACTER,
                       INPUT except-fields AS CHARACTER )

PUBLIC BindingSource ( INPUT dataset-hdl AS HANDLE )

PUBLIC BindingSource ( INPUT dataset-hdl AS HANDLE,
                     { INPUT parent-buffer-name AS CHARACTER
                     | INPUT parent-buffer-hdl AS HANDLE } )

PUBLIC BindingSource ( INPUT dataset-hdl AS HANDLE,
                       INPUT include-fields AS CHARACTER,
                       INPUT except-fields AS CHARACTER )

PUBLIC BindingSource ( INPUT dataset-hdl AS HANDLE,
                     { INPUT parent-buffer-name AS CHARACTER
                     | INPUT parent-buffer-hdl AS HANDLE },
                      INPUT include-fields AS CHARACTER,
                      INPUT except-fields AS CHARACTER )

PUBLIC BindingSource ( INPUT dataset-hdl AS HANDLE,
                       INPUT report-mode AS LOGICAL )

PUBLIC BindingSource ( INPUT dataset-hdl AS HANDLE,
                     { INPUT parent-buffer-name AS CHARACTER
                     | INPUT parent-buffer-hdl AS HANDLE },
                       INPUT include-fields AS CHARACTER,
                       INPUT except-fields AS CHARACTER,
                       INPUT report-mode AS LOGICAL )

PUBLIC BindingSource ( INPUT buffer-hdl AS HANDLE )

PUBLIC BindingSource ( INPUT buffer-hdl AS HANDLE,
                       INPUT include-fields AS CHARACTER,
                       INPUT except-fields AS CHARACTER )

PUBLIC BindingSource ( )
```

query-hdl

A HANDLE variable that represents the handle to a query object.

dataset-hdl

A HANDLE variable that represents the handle to a ProDataSet object.

buffer-hdl

A HANDLE variable that represents the handle to a buffer object.

parent-buffer-name

A CHARACTER variable that represents the name of a parent buffer in a ProDataSet object. This corresponds to the top-level table displayed in a hierarchical control (such as a grid or a treeview).

parent-buffer-hdl

A HANDLE variable that represents the handle to a parent buffer in a ProDataSet object. This corresponds to the top-level table displayed in a hierarchical control (such as a grid or a treeview).

include-fields

An optional CHARACTER expression that evaluates to a comma-separated list of fields in the ABL data source object to bind to the ProBindingSource. To include all fields, specify an asterisk (*) in *include-fields*. To include all but a few fields, specify an asterisk (*) in *include-fields* and a comma-separated list of fields to exclude in *except-fields*. If *include-fields* contains specific fields, *except-fields* is ignored.

When including fields for a ProDataSet object, you must specify a comma-separated list of fields to display in display order and qualify the field names with their buffer name. For example:

```
"table1.field1, table1.field2, table2.field1, table1.field3"
```

You must specify at least one field from each table in the hierarchy to be displayed.

When including fields for a query or a join query, you must specify a comma-separated list of fields to display in display order and you should qualify ambiguous field names with their buffer name.

Do not specify a RAW temp-table or a database field in *include-fields*. If you explicitly specify a RAW field, the AVM generates a run-time error. If you specify an asterisk (*) to include all fields and there is a RAW field in the table, the AVM excludes it automatically even if it is not specified in *except-fields*.

except-fields

A CHARACTER expression that evaluates to a comma-separated list of fields in the ABL data source object to exclude from binding to the ProBindingSource. You can specify fields to exclude only when *include-fields* contains an asterisk (*). If you do not have any fields to exclude, you must specify the empty string (""). If *include-fields* contains specific fields, *except-fields* is ignored.

report-mode

When *report-mode* is set to true, it indicates that any control that binds to this ProBindingSource will be a non-graphical or reporting object. This allows the ProBindingSource to make optimizations to reduce memory usage.

Binding to a query associated with a temp-table or database table

When binding to a temp-table or database table through a query, consider the following:

- The query may, or may not, be part of a ProDataSet object.
- The query can be either static or dynamic.
- You use a handle to access either a static or a dynamic query.
- The query must be scrolling. That is, a static query must be defined with the SCROLLING keyword and a dynamic query, which defaults to scrolling, must remain that way (its FORWARD-ONLY attribute must be FALSE).
- The query can be a join involving one or more tables. In this case, one row is displayed in the control for each result list entry.

- The ProBindingSource takes information from the table schema definition and makes it available to the bound .NET control for headings and labels. The ProBindingSource determines headings and labels by using either the column label (if defined), the field label (if defined), or the field name from the schema.

  If the query is a join with duplicate field names in the tables, you should qualify the field names with the buffer names using the *buffer-name*.*field-name* syntax. Also, the ProBindingSource treats elements in an array field as separate fields and makes them unique by appending their array subscript using the *array-name*[*subscript*] syntax.

- If the .NET control does not handle sorting on its own, you can do either of the following:

  - Let the ProBindingSource automatically sort records in the ABL data source object by setting the AutoSort property to TRUE.
  - Define a trigger to handle a SortRequest event.

  Note: If you need to reopen the query, you must do so using the QUERY-PREPARE and QUERY-OPEN methods (not the OPEN QUERY statement). Progress Software Corporation recommends that you reopen the query using the PRESELECT option, which optimizes query performance.

- The ProBindingSource automatically changes currency in the query (as reflected by the Position property) as the selected row in the bound .NET control changes.

Binding to a ProDataSet object

Binding to a ProDataSet object enables you to create a hierarchical display of a parent buffer and its child buffers in the same .NET control (such as a grid or a treeview). When binding to a ProDataSet object, consider the following:

- The ProDataSet object can contain one or more parent and child buffers (and a set of data relations describing the relationship between the parent and child buffers).
- The ProDataSet object can be either static or dynamic.
- You use a handle to access either a static or dynamic ProDataSet object.
- You can identify a temp-table in a ProDataSet object by either its name or its handle.
- To bind a hierarchy of parent and child buffers to a single .NET control, specify a parent buffer from the ProDataSet object as the top-level table to display in the .NET control. A parent buffer is typically a top-level buffer in the ProDataSet object, but it need not be. Any child buffers of the specified parent buffer are automatically included in the hierarchy. The ProBindingSource makes a buffer's data available to a bound .NET control as follows:

  - If you specify a parent buffer that is a top-level buffer, the ProBindingSource uses the buffer's top-level navigation query (specified by the TOP-NAV-QUERY attribute) to populate the primary set of records, and uses the data-relation queries associated with each of the child buffers for subsequent levels.
  - If you specify a parent buffer that is not a top-level buffer, the ProBindingSource uses the buffer's data-relation query to populate the primary set of records, and uses the data-relation queries associated with each of the child buffers for subsequent levels.
  - If you specify a buffer that is not a parent buffer (that is, a buffer that has no child buffers), the ProBindingSource binds to a single table. You can achieve this same result by binding to the buffer's top-level navigation query (specified by the TOP-NAV-QUERY attribute) or its data-relation query.
  - If you do not specify any buffer, the ProBindingSource binds to the first top-level buffer in the ProDataSet object. The first top-level buffer is based on the order in which the buffers were defined or added in the ProDataSet object, and it matches the result from GET-TOP-BUFFER(1).

- To display both parent and child buffers within the same .NET control, use a single ProBindingSource instance bound to the ProDataSet object. To display the parent and child buffers in separate .NET controls, use separate ProBindingSource instances bound to their respective queries.
- The ProBindingSource automatically changes currency in the parent buffer (as reflected by the Position property) as the selected parent or child row in the bound .NET control changes.

  For more information on how the ProBindingSource manages currency between parent and child buffers, including a code example, see the data-binding topics in *Use GUI for .NET Objects in ABL Applications*.
- When the user first expands (not selects) a parent row in the bound .NET control, all of the child rows for that parent row are displayed. The ProBindingSource uses the ProDataSet object's data-relation query to create and maintain a unique query for all child rows related to the expanded parent row. For example, if the user expands multiple parent rows, there will be an open child query for each expanded parent row based on the corresponding parent record.
- The ProBindingSource takes information from the ProDataSet object schema definition and makes it available to the bound .NET control for headings and labels. The ProBindingSource determines headings and labels by using either the column label (if defined), the field label (if defined), or the field name from the schema.

Binding to a buffer object

When binding to a temp-table or database buffer, consider the following:

- The buffer might (but not necessarily) be part of a ProDataSet object.
- The buffer can be either static or dynamic.
- You use a handle to access either a static or dynamic buffer.
- Use a buffer as a data source for one or more single-value controls (such as a check box or a text box).
- The Position property value will always be set to zero. Therefore, you must manage currency for the buffer by changing the record in the ABL data source object programmatically.

Binding to an unbound ProBindingSource instance

An unbound ProBindingSource instance is created by invoking the ProBindingSource constructor with no parameters. When binding to an unbound ProBindingSource instance, consider the following:

- To limit which fields are exposed in the data source object, use the `SetFields( )` method before binding to the data source object.
- You must bind an unbound ProBindingSource object instance to an ABL data source object at run time by setting the ProBindingSource's `Handle` property to the handle of the associated ABL data source object.
- When you set the ProBindingSource's Handle property, the AVM includes, by default, all fields from the ABL data source object's schema in the .NET control. (This is equivalent to specifying an asterisk (*) in the *`include-fields`* parameter when invoking a ProBindingSource constructor with parameters.) If you want to limit the available fields, you must use the ProBindingSource's `SetFields( )` method before you specify the Handle property. Attempting to use the `SetFields( )` method after specifying the `Handle` property causes the ProBindingSource to throw a .NET exception.

Super Class

`System.Windows.Forms.BindingSource` class (from the .NET Framework)

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

AllowEdit property	AllowNew property	AllowRemove property
AutoSort property	AutoSync property	AutoUpdate property
Batching property	ChildAllowEdit property	ChildAllowNew property
ChildAllowRemove property	ChildInputValue property	Count property
Handle property	InputValue property	MapNullToEmptyString property
MaxDataGuess property	NewRow property	Position property (BindingSource)
RowModified property	SyncToSelectedRow property	TableSchema property
Tag property (Progress.Data.BindingSource)	UseFullFieldNames property	—

Public Methods

Assign( ) method (Progress.Data.BindingSource)	Dispose( ) method
Refresh( ) method (Progress.Data.BindingSource)	RefreshAll( ) method
SetFields( ) method	—

Public Events

AfterFetchNextBatch event	BeforeFetchNextBatch event
CreateRow event	CancelCreateRow event
OffEnd event (.NET)	PositionChanged event
SortRequest event	—

Notes

- Since you are working with a combination of .NET controls and ABL objects, it is important to remember that .NET control parameter and property indexes are always zero-based values and ABL indexes are always one-based values.
- For information on the public class members inherited from the `System.Windows.Forms.BindingSource` class, refer to the .NET Framework class library.

Progress.Data.CancelCreateRowEventArgs class
(Windows only; GUI for .NET only)

A `CancelCreateRowEventArgs` object instance that is an argument to an event handler for a `CancelCreateRow` event.

Serializable:

No

Constructors

This class contains a private constructor; you cannot instantiate it directly.

Super Class

`System.EventArgs` class (from the .NET Framework)

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

BandIndex property	BufferHdl property	BufferName property

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events (beyond those it inherits from its base class).

Note

For information on the public class members inherited from the `System.EventArgs` class, refer to the .NET Framework class library.

See also

CancelCreateRow event, CreateRow event, CURRENT-QUERY( ) method, Progress.Data.CreateRowEventArgs class

---

Progress.Data.ChildrenAllowEdit class
(Windows only; GUI for .NET only)

`ChildrenAllowEdit` is an internal class used for the implementation of the ChildAllowEdit property of the Progress.Data.BindingSource class. There is no need to use `ChildrenAllowEdit` directly.

Progress.Data.ChildrenAllowNew class
(Windows only; GUI for .NET only)

`ChildrenAllowNew` is an internal class used for the implementation of the ChildAllowNew property of the Progress.Data.BindingSource class. There is no need to use `ChildrenAllowNew` directly.

Progress.Data.ChildrenAllowRemove class
(Windows only; GUI for .NET only)

`ChildrenAllowRemove` is an internal class used for the implementation of the ChildAllowRemove property of the Progress.Data.BindingSource class. There is no need to use `ChildrenAllowRemove` directly.

Progress.Data.ChildrenInputValue class
(Windows only; GUI for .NET only)

`ChildrenInputValue` is an internal class used for the implementation of the ChildInputValue property of the Progress.Data.BindingSource class. There is no need to use `ChildrenInputValue` directly.

Progress.Data.ColumnPropDesc class
(Windows only; GUI for .NET only)

A design-time object that the Visual Designer uses to describe a table column (field) as part of logical schema information that it associates with a Progress.Data.BindingSource class instance using the Progress.Data.TableDesc class.

Note: This class is designed for use by the Visual Designer. Though you can do so, it is not expected that you will use this in your own code.

Serializable:

No

Constructor

```
PUBLIC ColumnPropDesc ( INPUT col-name AS CHARACTER ,
                        INPUT col-label AS CHARACTER ,
                        INPUT col-type AS CLASS Progress.Data.DataType )
```

col-name

A CHARACTER expression that represents the name of the table column.

col-label

A CHARACTER expression that represents a label for the table column. If *col-label* is set to the Unknown value (?), the label for the field is determined at run time according to OpenEdge column label rules.

Note: A field label specified in a control always takes precedence over the field label supplied by the BindingSource.

col-type

A Progress.Data.DataType enumeration class value that describes the ABL data type of the column.

Super Class

`System.ComponentModel.PropertyDescriptor` class (from the .NET Framework)

Interfaces

`System.Runtime.Serialization.ISerializable` interface (from the .NET Framework)

Public Properties

The public properties defined by this class are not supported for use in applications.

Public Methods

The public methods defined by this class are not supported for use in applications.

Public Events

The public events defined by this class are not supported for use in applications.

Note

For information on the public class members inherited from the `System.ComponentModel.PropertyDescriptor` class, refer to the .NET Framework class library.

See also

Progress.Data.BindingSource class, Progress.Data.DataType enumeration class, Progress.Data.TableDesc class

---

Progress.Data.CreateRowEventArgs class
(Windows only; GUI for .NET only)

A `CreateRowEventArgs` object instance that is an argument to an event handler for a `CreateRow` event.

Serializable:

No

Constructors

This class contains a private constructor; you cannot instantiate it directly.

Super Class

`System.EventArgs` class (from the .NET Framework)

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

BandIndex property	BufferHdl property	BufferName property
Created property	–	–

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events (beyond those it inherits from its base class).

Note

For information on the public class members inherited from the `System.EventArgs` class, refer to the .NET Framework class library.

See also

CancelCreateRow event, CreateRow event, CURRENT-QUERY( ) method, Progress.Data.CancelCreateRowEventArgs class

---

Progress.Data.DataType enumeration class
(Windows only; GUI for .NET only)

An enumeration whose members indicate the ABL built-in data types that are valid for use with the Progress.Data.BindingSource class (the ProBindingSource).

Members

Member	Description
BLOB	Indicates the BLOB data type
CHARACTER	Indicates the CHARACTER data type
CLOB	Indicates the CLOB data type
COM_HANDLE	Indicates the COM-HANDLE data type
DATE	Indicates the DATE data type
DATETIME	Indicates the DATETIME data type
DATETIME_TZ	Indicates the DATETIME-TZ data type
DECIMAL	Indicates the DECIMAL data type
HANDLE	Indicates the HANDLE data type
INT64	Indicates the INT64 data type

Member	Description
INTEGER	Indicates the INTEGER data type
LOGICAL	Indicates the LOGICAL data type
PROGRESS_LANG_OBJECT	Indicates the Progress.Lang.Object class type
RECID	Indicates the RECID data type
ROWID	Indicates the ROWID data type
UNKNOWN	Data type unknown (specified only at run time)

Super Class

`System.Enum` class (from the .NET Framework)

Notes

- For information on the System.Enum class, refer to the .NET Framework class library.
- The ProBindingSource exposes an ABL BLOB field (a Binary Large Object) to any bound .NET control as a .NET `System.Byte[]` data type. Before binding a BLOB to a particular .NET control, check the .NET control's user documentation to see if it can render the type of file that the BLOB represents.

  For example, if you bind an image BLOB to a Microsoft `DataGridView`, the grid represents the BLOB as a `Systems.Windows.Forms.DataGridViewImageColumn` class. If the BLOB is not in a recognized format, the class throws an error. If you bind the same BLOB to an `UltraGrid` from the OpenEdge Advanced UI Controls, the grid represents the BLOB as an `UltraGridColumn` of the `System.Byte[]` data type. In order to display the image, you must then add an editor control to the column using something like the following code:

  ```
  myGrid:DisplayLayout:Bands[0]:Columns[1]:Editor =
    NEW Infragistics.Win.EmbeddableImageRender().
  ```

- The ProBindingSource exposes an ABL CLOB field (a Character Large Object) to any bound .NET control as a .NET `System.String` data type. Be sure that the .NET control can handle that data type.

See also

Progress.Data.ColumnPropDesc class

Progress.Data.DummyRow class
(Windows only; GUI for .NET only)

Controls that do data binding need to be able to access rows of the data source they are bound to. Given the row, they can then ask for column values to display on the screen or to manipulate. Because real temp-table rows are not accessible to .NET, the `ProBindingSource` creates row objects in .NET of type `Progress.Data.DummyRow`. When the control asks for data from one of these rows, the `ProBindingSource` can then get the appropriate data from the AVM.

Normally you do not need to, and should not, interact with this class. However, methods of some controls will return an instance of the `Progress.Data.DummyRow` class. The only property of interest would be the `Index` property. This is equivalent to the row number in the temp-table. For example, you could use it to do `REPOSITION TO ROW` *`index`*`.`

Serializable:

No

Constructor(s)

This class contains a private constructor; you cannot instantiate it directly.

Super Class

None

Interfaces

None

Public Properties

Index	–

Public Methods

This class does not contain methods.

Public Events

This class does not contain events.

See also

Progress.Data.BindingSource class

Progress.Data.InputValue class
(Windows only; GUI for .NET only)

Provides indexers you can use to access the input value of the specified field in the current row of a table displayed in the bound .NET control.

Note: Using an instance of this class is analogous to using the INPUT-VALUE attribute for a field value displayed in an ABL widget.

Serializable:

No

Constructors

This class does not contain a constructor.

Super Class

`System.Object` class (from the .NET Framework)

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

This class does not contain properties (beyond those it inherits from its base class).

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events (beyond those it inherits from its base class).

Notes

- The following code fragment illustrates how to use this class:

```
DEFINE VARIABLE myCustName AS CHARACTER NO-UNDO.

/* InputValue indexer is 0-based */
myCustName = UNBOX(myBindingSource:InputValue[1]).
IF myCustName EQ "Brown" THEN DO:
  MESSAGE "Invalid Customer Name".
  . . .
END.
```

- For information on the public class members inherited from the `System.Object` class, refer to the .NET Framework class library.

See also

ChildInputValue property, InputValue property

Progress.Data.OffEndEventArgs class
(Windows only; GUI for .NET only)

An `OffEndEventArgs` object instance that is an argument to an event handler for an `OffEnd` event.

Serializable:

No

Constructors

This class contains a private constructor; you cannot instantiate it directly.

Super Class

`System.EventArgs` class (from the .NET Framework)

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

RowsAdded property	–	–

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events (beyond those it inherits from its base class).

Note

For information on the public class members inherited from the `System.EventArgs` class, refer to the .NET Framework class library.

See also

Batching property, OffEnd event (.NET)

Progress.Data.SortRequestEventArgs class
(Windows only; GUI for .NET only)

A `SortRequestEventArgs` object instance that is an argument to an event handler for a `SortRequest` event.

Serializable:

No

Constructors

This class contains a private constructor; you cannot instantiate it directly.

Super Class

`System.EventArgs` class (from the .NET Framework)

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

ArrayIndex property	Ascending property	BufferHdl property
FieldIndex property	FieldName property	Sorted property

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events (beyond those it inherits from its base class).

Note

For information on the public class members inherited from the `System.EventArgs` class, refer to the .NET Framework class library.

See also

SortRequest event

Progress.Data.TableDesc class
(Windows only; GUI for .NET only)

A design-time object that the Visual Designer uses to store logical schema information that you specify for the data source associated with a Progress.Data.BindingSource class instance. The specified logical schema can represent a single temp-table or a hierarchy of temp-tables in a ProDataSet, where the current instance describes the parent for a specified set of child tables.

Note: This class is designed for use by the Visual Designer. Though you can do so, it is not expected that you will use this in your own code.

Serializable:

No

Constructor

```
PUBLIC TableDesc ( INPUT name AS CHARACTER ,
                   INPUT columns AS CLASS "Progress.Data.ColumnPropDesc[]" ,
                   INPUT children AS CLASS "Progress.Data.TableDesc[]" )
```

name

A CHARACTER expression that represents a table name for the logical schema.

columns

An array of Progress.Data.ColumnPropDesc class instances that describe the fields of the table represented by this TableDesc instance.

children

An array of Progress.Data.TableDesc class instances that describe child tables of the parent table represented by this instance. If this table has no child tables, *children* is set to the Unknown value (?).

Super Class

`System.Object` class (from the .NET Framework)

Interfaces

`System.Runtime.Serialization.ISerializable` interface (from the .NET Framework).

Public Properties

The public properties defined by this class are not supported for use in applications.

Public Methods

The public methods defined by this class are not supported for use in applications.

Public Events

The public events defined by this class are not supported for use in applications.

Note

For information on the public class members inherited from the `System.Object` class, refer to the .NET Framework class library.

See also

Progress.Data.BindingSource class, Progress.Data.ColumnPropDesc class, TableSchema property

---

Progress.Database.TempTableInfo class

This class provides information about a temp-table and its index.

Serializable:

No

Constructors

This class contains only a private constructor. You cannot instantiate it directly.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

ArchiveIndexStatistics property	ArchiveTableStatistics property	TempTableCount property
TempTablePeak property	–	–

Public Methods

GetIndexInfoByID( ) method	GetIndexStatHistoryHandle( ) method
GetTableInfoByID( ) method	GetTableInfoByPosition( ) method
GetTableStatHistoryHandle( ) method	GetVSTHandle( ) method

Public Events

This class does not contain events.

Notes

For more information on using this class, see *Troubleshoot ABL Applications*.

Progress.Database.VSTTableId class

This class identifies the Virtual System Table (VST) ID when you need to access a dynamic VST temp-table.

Serializable:

No

Constructors

This class contains a private constructor. You cannot instantiate it directly.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Methods

This class does not contain any methods.

Public Properties

ActBufferId property	ActIndexId property	ActIOFileId property
ActIOTypeId property	ActOtherId property	ActRecordId property
ActSpaceId property	ActSummaryId property	BlockId property

BuffStatusId property	DbStatusId property	FileListId property
IndexStatId property	MstrBlkId property	StatBaseId property
TableStatId property	TransId property	UserIOId property
UserIndexStatId property	UserTableStatId property	–

Public Events

This class does not contain events.

Progress.IO.BinarySerializer class

`Progress.IO.BinarySerializer` provides functionality to serialize and deserialize ABL class-based objects in binary format. The serialization and deserialization are performed in conjunction with classes that inherit from `Progress.IO.InputStream` and `Progress.IO.OutputStream`. `Progress.IO.BinarySerializer` formats the data, and the stream classes determine where the data comes from and goes to. For example, `Progress.IO.FilleInputStream` and `Progress.IO.FileOutputStream` are used for serialization to and deserialization from a file.

Serializable:

No

Constructors

```
PUBLIC BinarySerializer ( )
```

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces.

Properties

This class does not define properties.

Public Methods

Deserialize( ) method (BinarySerializer)	Serialize( ) method (BinarySerializer)

Public Events

This class does not contain events.

Notes

- See the entry for the `Serialize( )` method for more information about which class members are serialized.

See also

Progress.IO.FileInputStream class, Progress.IO.FileOutputStream class, Progress.IO.InputStream class, Progress.IO.OutputStream class

Progress.IO.ByteOrder enumeration

An enumeration to support the MemoryOutputStream object's `ByteOrder` enum-type property.

Members

Member	Description
`HostByteOrder`	The byte order employed by the machine where the process is running.
`BigEndian`	Byte ordering where the most significant value in a sequence is stored first (i.e.in the lowest address). Successive bytes are stored at successively higher addresses. For example, the two bytes required for the hexadecimal number 9A62 would be stored as 9A in the first address space and 62 in the second when byte ordering is big-endian. Note that Internet protocols use big-endian byte-ordering.
`LittleEndian`	Byte ordering where the least significant value in a sequence is stored first (i.e.in the lowest address). Successive bytes are stored at successively higher addresses. For example, the two bytes required for the hexadecimal number 9A62 would be stored as 62 in the first address space and 9A in the second when byte ordering is little-endian.

Progress.IO.FileInputStream class

`Progress.IO.FileInputStream` extends `Progress.IO.InputStream` and provides functionality for reading from a file.

Serializable:

No

Constructor

```
PUBLIC FileInputStream ( INPUT filename AS CHARACTER )
```

filename

A CHARACTER expression that represents the file from which the input stream will read. The constructor sets the FileName property of the stream to *filename*. The file name can include Unicode characters, and it can be absolute or relative to the current working directory. The AVM raises a runtime error if it can't open the file for reading.

Super Class

Progress.IO.InputStream class

Interfaces

This class does not implement an interface.

Public Properties

FileName property	–

Public Methods

Close( ) method (IO)	SkipBytes( ) method (FileInputStream)

Public Events

This class does not define events.

Progress.IO.FileOutputStream class

Progress.IO.FileOutputStream extends Progress.IO.OutputStream and provides functionality for writing to a file.

Serializable:

No

Constructor

```
PUBLIC FileOutputStream ( INPUT filename AS CHARACTER )

PUBLIC FileOutputStream ( INPUT filename AS CHARACTER,
                          INPUT append AS LOGICAL )
```

append

A LOGICAL expression that indicates whether the output stream can be appended to. The constructor sets the Append property of the stream object to the value of *append*.

filename

A CHARACTER expression that represents the file to which the output stream will be written. The constructor sets the FileName property of the stream object to *filename*. The file name can include

Unicode characters, and it can be absolute or relative to the current working directory. The AVM raises a runtime error if it can't open the file for writing.

Super Class

Progress.IO.OutputStream class

Interfaces

This class does not implement an interface.

Public Properties

Append property	FileName property

Public Methods

Close( ) method (IO)	Flush( ) method (FileOutputStream)

Public Events

This class does not define events.

Progress.IO.InputStream class

`Progress.IO.InputStream` is an abstract class that provides the default implementation for all input stream classes. For example, `Progress.IO.InputStream` is the class from which the `Progress.IO.FileInputStream` class is derived.

Serializable:

No

Constructor

This class contains a private constructor; as an abstract class, you cannot instantiate it directly.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement an interface.

Public Properties

Closed property	–

Public Methods

Close( ) method (IO)	Read( ) method (InputStream)
SkipBytes( ) method (InputStream)	–

Public Events

This class does not define events.

Progress.IO.JsonSerializer class

`Progress.IO.JsonSerializer` provides functionality to serialize and deserialize ABL class-based objects in JSON format. The serialization and deserialization are performed in conjunction with classes that inherit from `Progress.IO.InputStream` and `Progress.IO.OutputStream`. `Progress.IO.JsonSerializer` formats the data, and the stream classes determine where the data comes from and goes to. For example, `Progress.IO.FilleInputStream` and `Progress.IO.FileOutputStream` are used for serialization to and deserialization from a file.

Serializable:

No

Constructors

```
PUBLIC JsonSerializer ( INPUT formatted AS LOGICAL )
```

formatted

When formatted is TRUE, the JSON string is formatted in a hierarchical manner using white space, carriage returns, and line feeds.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces.

Properties

Formatted property	–

Public Methods

Deserialize( ) method (JsonSerializer)	Serialize( ) method (JsonSerializer)

Public Events

This class does not contain events.

Notes

- See the entry for the `Serialize( )` method for more information about which class members are serialized and the format of the object serialization.

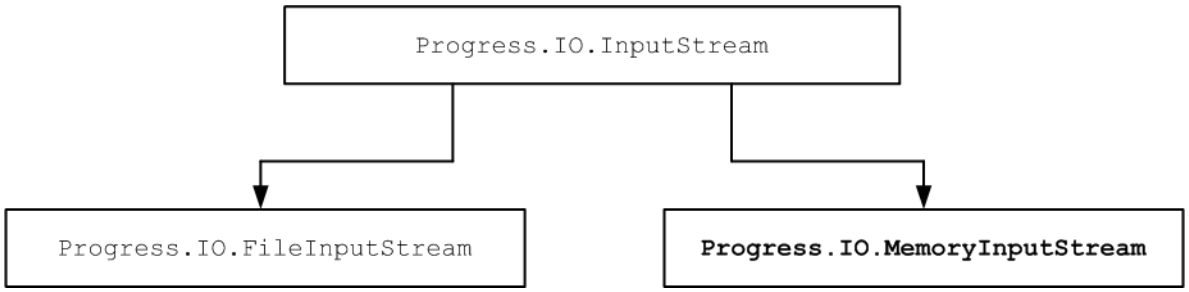
See also

Progress.IO.FileInputStream class, Progress.IO.FileOutputStream class, Progress.IO.InputStream class, Progress.IO.OutputStream class

Progress.IO.MemoryInputStream class

This class is responsible for reading data from memory. It acts as a wrapper for MEMPTR read operations.

`Progress.IO.MemoryInputStream` extends `Progress.IO.InputStream`:



As with other input stream classes, the `MemoryInputStream` class maintains its own internal read position and eliminates the need to keep track of MEMPTR read positions.

For input streams the expected network byte order is controlled by the underlying MEMPTR byte ordering. If the input stream was created using `MemoryOutputStream`, then the byte order is taken from the value of the ByteOrder property of the `MemoryOutputStream`. If instead the input stream was created from a MEMPTR, the SET-BYTE-ORDER statement must be used to control the byte ordering used by the `MemoryInputStream` read methods.

Serializable:

No

Constructor

```
PUBLIC MemoryInputStream ( INPUT mos AS Progress.IO.MemoryOutputStream )
PUBLIC MemoryInputStream ( INPUT mem AS MEMPTR )
```

mos

An initialized `MemoryOutputStream` object that this input stream object will operate on.

mem

An initialized MEMPTR variable that this input stream object will operate on.

Super Class

Progress.IO.InputStream class

Interfaces

This class does not implement an interface.

Public Properties

Data property	Position property (MemoryInputStream)

Public Methods

AvailableBytes( ) method	Read( ) method (MemoryInputStream)
ReadByte( ) method	ReadDouble( ) method
ReadFloat( ) method	ReadInt64( ) method
ReadLong( ) method	ReadLongchar( ) method
ReadShort( ) method	ReadString( ) method
ReadUnsignedLong( ) method	ReadUnsignedShort( ) method
Reset( ) method (MemoryInputStream)	SkipBytes( ) method (MemoryInputStream)

Public Events

This class does not define events.

Examples

Table 128: Read and write integer and decimal methods

```
DEFINE VARIABLE myDate AS DATE INITIAL "1/1/2020".
DEFINE VARIABLE myInt64 AS INT64 INITIAL 42.
DEFINE VARIABLE myDouble AS DECIMAL INITIAL 42.42.
DEFINE VARIABLE myFloat AS DECIMAL INITIAL 42.42.
DEFINE VARIABLE myLong AS INTEGER INITIAL -53.
DEFINE VARIABLE myShort AS INTEGER INITIAL -53.
DEFINE VARIABLE myULong AS INTEGER INITIAL 123456789.
DEFINE VARIABLE myUShort AS INTEGER INITIAL 12345.
DEFINE VARIABLE inMs AS Progress.IO.MemoryInputStream.
DEFINE VARIABLE outMs AS Progress.IO.MemoryOutputStream.

outMs = NEW Progress.IO.MemoryOutputStream().
outMs:WriteLong(INTEGER(myDate)).
outMs:WriteInt64(myInt64).
outMs:WriteDouble(myDouble).
outMs:WriteFloat(myFloat).
outMs:WriteLong(myLong).
outMs:WriteShort(myShort).
outMs:WriteUnsignedLong(myULong).
outMs:WriteUnsignedShort(myUShort).

inMs = NEW Progress.IO.MemoryInputStream(INPUT outMs:Data).
MESSAGE
  "ReadLong:" inMs:ReadLong() SKIP
  "ReadInt64:" inMs:ReadINT64() SKIP
  "ReadDouble:" inMs:ReadDouble() SKIP
  "ReadFloat:" inMs:ReadFloat() SKIP
  "ReadLong:" inMs:ReadLong() SKIP
  "ReadShort:" inMs:ReadShort() SKIP
  "ReadUnsignedLong:" inMs:ReadUnsignedLong() SKIP
  "ReadUnsignedShort:" inMs:ReadUnsignedShort() SKIP
  VIEW-AS ALERT-BOX.
```

Table 129: Read and write string methods

```
DEFINE VARIABLE outMs AS Progress.IO.MemoryOutputStream.
DEFINE VARIABLE InMs1 AS Progress.IO.MemoryInputStream.
DEFINE VARIABLE InMs2 AS Progress.IO.MemoryInputStream.

outMs = NEW Progress.IO.MemoryOutputStream().
outMs:Write("PROGRESS1",YES).
outMs:Write("PROGRESSx2",YES).

InMs1 = NEW Progress.IO.MemoryInputStream(INPUT outMs).
MESSAGE InMs1:ReadString()
  VIEW-AS ALERT-BOX.
MESSAGE STRING(InMs1:ReadLongchar())
  VIEW-AS ALERT-BOX.
```

See also

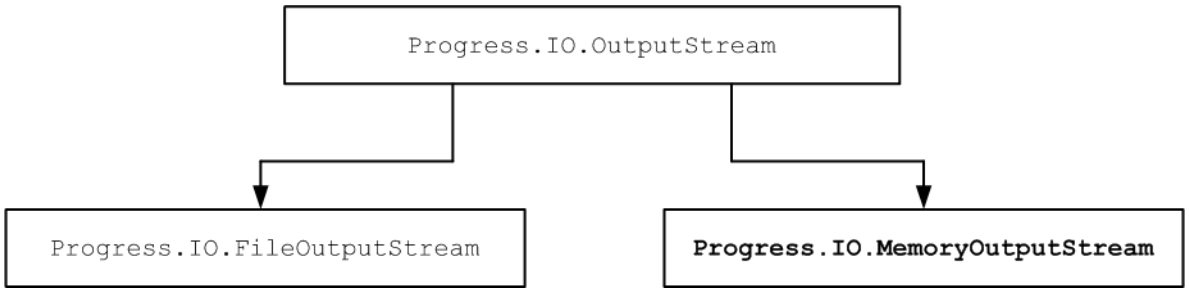
OpenEdge Programming Interfaces for more information about MEMPTR.

Progress.IO.MemoryOutputStream class

Progress.IO.MemoryOutputStream class

This class is responsible for streaming data to memory. A `MemoryOutputStream` object enables write operations to memory and is expandable. It encapsulates the MEMPTR data type and simplifies MEMPTR operations.

`Progress.IO.MemoryOutputStream` extends `Progress.IO.OutputStream`:



As with other output stream classes, the `MemoryOutputStream` class maintains its own internal write position and eliminates the need to keep track of MEMPTR write positions.

Serializable

No

Constructors

```
PUBLIC MemoryOutputStream ( )
PUBLIC MemoryOutputStream (INPUT size AS INT64)
```

size

The `Data` and `Length` properties are initialized to the value specified by *size*.

When *size* is not specified, the `Data` and `Length` properties are initialized to `0`.

Super Class

Progress.IO.OutputStream class

Interfaces

This class does not implement an interface.

Public Properties

ByteOrder property	BytesWritten property
Data property	Length property (MemoryOutputStream)

Public Methods

Clear( ) method (MemoryOutputStream)	SetByteOrder( ) method

Write( ) method (MemoryOutputStream)	WriteByte( ) method
WriteDouble( ) method	WriteInt64( ) method
WriteFloat( ) method	WriteLong( ) method
WriteShort( ) method	WriteUnsignedLong( ) method
WriteUnsignedShort( ) method	–

Public Events

This class does not define events.

Note

The following table illustrates values that result from constructors and `Write( )`/`Clear( )` methods:

Method/Constructor	**Values**	
	Obj:Length	**Obj:BytesWritten**
`Obj = NEW MemoryOutputStream()`	0	0
`Obj:Write("12345")`	5	5
`Obj:Clear()`	0	0
`Obj = NEW MemoryOutputStream(100)`	100	0
`Obj:Write("12345678")`	100	8
`Obj:Clear()`	100	0

Examples

Table 130: READ & WRITE decimal and integer methods

```
DEFINE VARIABLE myDate AS DATE INITIAL "1/1/2020".
DEFINE VARIABLE myInt64 AS INT64 INITIAL 42.
DEFINE VARIABLE myDouble AS DECIMAL INITIAL 42.42.
DEFINE VARIABLE myFloat AS DECIMAL INITIAL 42.42.
DEFINE VARIABLE myLong AS INTEGER INITIAL -53.
DEFINE VARIABLE myShort AS INTEGER INITIAL -53.
DEFINE VARIABLE myULong AS INTEGER INITIAL 123456789.
DEFINE VARIABLE myUShort AS INTEGER INITIAL 12345.
DEFINE VARIABLE inMs AS Progress.IO.MemoryInputStream.
DEFINE VARIABLE outMs AS Progress.IO.MemoryOutputStream.

outMs = NEW Progress.IO.MemoryOutputStream().
outMs:WriteLong(INTEGER(myDate)).
outMs:WriteInt64(myInt64).
outMs:WriteDouble(myDouble).
outMs:WriteFloat(myFloat).
outMs:WriteLong(myLong).
outMs:WriteShort(myShort).
outMs:WriteUnsignedLong(myULong).
outMs:WriteUnsignedShort(myUShort).

inMs = NEW Progress.IO.MemoryInputStream(INPUT outMs:Data).
MESSAGE
  "ReadLong:" inMs:ReadLong() SKIP
  "ReadInt64:" inMs:ReadINT64() SKIP
  "ReadDouble:" inMs:ReadDouble() SKIP
  "ReadFloat:" inMs:ReadFloat() SKIP
  "ReadLong:" inMs:ReadLong() SKIP
  "ReadShort:" inMs:ReadShort() SKIP
  "ReadUnsignedLong:" inMs:ReadUnsignedLong() SKIP
  "ReadUnsignedShort:" inMs:ReadUnsignedShort() SKIP
  VIEW-AS ALERT-BOX.
```

Table 131: READ & WRITE string methods

```
DEFINE VARIABLE outMs AS Progress.IO.MemoryOutputStream.
DEFINE VARIABLE InMs1 AS Progress.IO.MemoryInputStream.
DEFINE VARIABLE InMs2 AS Progress.IO.MemoryInputStream.

outMs = NEW Progress.IO.MemoryOutputStream().
outMs:Write("PROGRESS1",YES).
outMs:Write("PROGRESSx2",YES).

InMs1 = NEW Progress.IO.MemoryInputStream(INPUT outMs).
MESSAGE InMs1:ReadString()
  VIEW-AS ALERT-BOX.
MESSAGE STRING(InMs1:ReadLongchar())
  VIEW-AS ALERT-BOX.
```

See also

OpenEdge Programming Interfaces for more information about MEMPTR.

Progress.IO.MemoryInputStream class

Progress.IO.OutputStream class

`Progress.IO.OutputStream` is an abstract class that provides the default implementation for all output stream classes. For example, `Progress.IO.OutputStream` is the class from which the `Progress.IO.FileOutputStream` class is derived.

Serializable:

No

Constructor

This class contains a private constructor; as an abstract class, you cannot instantiate it directly.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement an interface.

Public Properties

Closed property	–

Public Methods

Close( ) method (IO)	Flush( ) method (OutputStream)
Write( ) method (OutputStream)	–

Public Events

This class does not define events.

Notes

- All of the overloads of the `Write( )` method are defined as abstract except for `Write( INPUT` *`target`* `AS MEMPTR )`. See the Write( ) method (OutputStream) entry for more information.

Progress.Json.JsonError class

This class raises an error when a failure specific to `Progress.Json` family of packages occur. For example, a `JsonError` is raised when a parameter is out of range or a reference is made to a non-existent property in a `JsonObject`.

Serializable:

Yes

Constructor

This class contains only a private constructor; you cannot instantiate it directly.

Super Class

Progress.Lang.SysError class

Interfaces

This class does not implement an interface.

Public Properties

This class does not define properties.

Public Methods

This class does not define methods.

Public Events

This class does not contain events.

Progress.Json.JsonParser class

This is an abstract class that contains properties common to JSON parsers supported by ABL.

Serializable:

No

Constructors

This class contains a private constructor; as an abstract class, you cannot instantiate it directly.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement an interface.

Public Properties

IgnoreComments property	–	–

Public Methods

This class does not define any methods.

Public Events

This class does not define any events.

Progress.Json.JsonParserError class

This class raises an error when the syntax of the JSON being parsed is incorrect.

Serializable:

Yes

Constructor

This class contains only a private constructor; you cannot instantiate it directly.

Super Class

Progress.Json.JsonError class

Interfaces

This class does not implement an interface.

Public Properties

Offset property	–	–

Public Methods

This class does not define methods.

Public Events

This class does not define events.

Progress.Json.ObjectModel.JsonArray class

This class denotes an array containing a dynamic number of heterogeneous elements addressable by a numeric index. Since the number of elements is dynamic, the array can grow or shrink in size. Elements within the array can be another `JsonArray`, a `JsonObject`, a JSON value of any data type or the JSON Null value. JSON data types are `String`, `Number` and `Boolean`. Indexing into `JsonArrays` is 1-based.

A `JsonArray` is heterogeneous because elements within the array may be of different data types. For example, an array may contain mostly numbers, but may contain some null values to indicate the absence of a value and may, contain strings to indicate special cases.

Serializable:

Yes

Constructors

```
PUBLIC JsonArray( )

PUBLIC JsonArray( INPUT initial-size AS INTEGER )

PUBLIC JsonArray( INPUT initial-value AS CHARACTER EXTENT )

PUBLIC JsonArray( INPUT initial-value AS COM-HANDLE EXTENT )

PUBLIC JsonArray( INPUT initial-value AS DATE EXTENT )

PUBLIC JsonArray( INPUT initial-value AS DATETIME EXTENT )

PUBLIC JsonArray( INPUT initial-value AS DATETIME-TZ EXTENT )

PUBLIC JsonArray( INPUT initial-value AS DECIMAL EXTENT )

PUBLIC JsonArray( INPUT initial-value AS HANDLE EXTENT )

PUBLIC JsonArray( INPUT initial-value AS INT64 EXTENT )

PUBLIC JsonArray( INPUT initial-value AS INTEGER EXTENT )

PUBLIC JsonArray( INPUT initial-value AS LOGICAL EXTENT )

PUBLIC JsonArray( INPUT initial-value AS LONGCHAR EXTENT )

PUBLIC JsonArray( INPUT initial-value AS MEMPTR EXTENT )

PUBLIC JsonArray( INPUT initial-value AS RAW EXTENT )

PUBLIC JsonArray( INPUT initial-value AS RECID EXTENT )

PUBLIC JsonArray( INPUT initial-value AS ROWID EXTENT )

PUBLIC JsonArray( INPUT initial-value AS JsonObject EXTENT )

PUBLIC JsonArray( INPUT initial-value AS JsonArray EXTENT )
```

initial-size

An INTEGER variable that represents the size of the new `JsonArray`. Each element is initialized to the JSON null value. If this value is 0, the array is empty. If this value is less than 0 or Unknown (?), a `JsonError` error is raised.

initial-value

An ABL array that contains values that are used to initialize elements in the new `JsonArray`. Each element of the ABL array is used to initialize one element in the `JsonArray`. The constructed `JsonArray` has as many elements as the length of the ABL array.

Any Unknown (?) values within the ABL array results in a JSON null value in the `JsonArray`. If initial-value is Unknown (?), or is indeterminate, a `JsonError` is raised.

Super Class

Progress.Json.ObjectModel.JsonConstruct class

Interfaces

This class does not implement an interface.

Public Properties

END_OF_ARRAY property	Length property (JSON)	–

Public Methods

Add( ) method (JsonArray)	AddNull( ) method (JsonArray)
AddNumber( ) method (JsonArray)	Clone( ) method (JsonArray)
GetCharacter( ) method (JsonArray)	GetCOMHandle( ) method (JsonArray)
GetDate( ) method (JsonArray)	GetDatetime( ) method (JsonArray)
GetDatetimeTZ( ) method (JsonArray)	GetDecimal( ) method (JsonArray)
GetHandle( ) method (JsonArray)	GetInt64( ) method (JsonArray)
GetInteger( ) method (JsonArray)	GetJsonArray( ) method (JsonArray)
GetJsonObject( ) method (JsonArray)	GetJsonText( ) method (JsonArray)
GetLogical( ) method (JsonArray)	GetLongchar( ) method (JsonArray)
GetMemptr( ) method (JsonArray)	GetRaw( ) method (JsonArray)
GetRecid( ) method (JsonArray)	GetRowid( ) method (JsonArray)
GetType( ) method (JsonArray)	IsNull( ) method (JsonArray)
Read( ) method (JsonArray)	Remove( ) method (JsonArray)
Set( ) method (JsonArray)	SetNull( ) method (JsonArray)
SetNumber( ) method (JsonArray)	WriteFile( ) method (JsonArray)
WriteFile( ) method (JsonArray)	WriteStream( ) method (JsonArray)

Public Events

This class does not contain events.

Progress.Json.ObjectModel.JsonConstruct class

`JsonConstruct` is an abstract class from which `JsonArray` class and `JsonObject` class are derived.

Serializable:

No

Constructor

This class contains only a private constructor; you cannot instantiate it directly.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement an interface.

Public Properties

This class does not define properties.

Public Methods

Write( ) method (JsonConstruct)	WriteFile( ) method (JsonConstruct)
WriteStream( ) method (JsonConstruct)	–

Public Events

This class does not define events.

Progress.Json.ObjectModel.JsonDataType class

Indicates the data type of a value in the JSON string. This class is returned by the GetType( ) method on a JsonObject or JsonArray.

Serializable:

No

Constructor

This class contains only a private constructor; you cannot instantiate it directly.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement an interface.

Public Properties

Array property	String property	Number property
Boolean property	Object property	Null property

Public Methods

This class does not define methods.

Public Events

This class does not contain events.

Progress.Json.ObjectModel.JsonObject class

This class denotes a dynamic number of properties each addressable by a Unicode string called a name. The properties can be another `JsonObject`, a `JsonArray`, a value of any JSON data type or the JSON null value. JSON data types are `String`, `Number` and `Boolean`.

Serializable:

Yes

Constructors

```
PUBLIC JsonObject( )
```

Interfaces

This class does not implement an interface.

Public Properties

This class does not define properties.

Public Methods

Add( ) method (JsonObject)	AddNull( ) method (JsonObject)
AddNumber( ) method (JsonObject)	Clone( ) method (JsonObject)
GetCharacter( ) method (JsonObject)	GetCOMHandle( ) method (JsonObject)
GetDate( ) method (JsonObject)	GetDatetime( ) method (JsonObject)
GetDatetimeTZ( ) method (JsonObject)	GetDecimal( ) method (JsonObject)
GetHandle( ) method (JsonObject)	GetInt64( ) method (JsonObject)
GetInteger( ) method (JsonObject)	GetJsonArray( ) method (JsonObject)
GetJsonObject( ) method (JsonObject)	GetJsonText( ) method (JsonObject)
GetLogical( ) method (JsonObject)	GetLongchar( ) method (JsonObject)
GetMemptr( ) method (JsonObject)	GetNames( ) method (JsonObject)

GetRaw( ) method (JsonObject)	GetRecid( ) method (JsonObject)
GetRowid( ) method (JsonObject)	GetType( ) method (JsonObject)
Has( ) method (JsonObject)	IsNull( ) method (JsonObject)
Read( ) method (JsonObject)	Remove( ) method (JsonObject)
Set( ) method (JsonObject)	SetNull( ) method (JsonObject)
SetNumber( ) method (JsonObject)	Write( ) method (JsonObject)
WriteFile( ) method (JsonObject)	WriteStream( ) method (JsonObject)

Public Events

This class does not define events.

Notes

The following example shows the constructor for `JsonObject`:

```
DEFINE VARIABLE myObj AS JsonObject NO-UNDO.
DEFINE VARIABLE myLongchar AS LONGCHAR NO-UNDO.

myObj = NEW JsonObject( ).
myObj:Write(myLongchar, TRUE).
```

In the above example, `myObj` is an empty object. The variable `myLongchar` is set to the value of `"{ }"`.

Progress.Json.ObjectModel.ObjectModelParser class

This class is used to parse multiple sources of JSON streams.

Serializable:

Yes

Constructors

```
PUBLIC ObjectModelParser( )
```

Super Class

Progress.Json.JsonParser class

Interfaces

This class does not implement an interface.

Public Properties

This class does not define properties.

Public Methods

Parse( ) method	ParseFile( ) method

Public Events

This class does not contain events.

Notes

The following example creates a parser and then parses JSON from the WebSpeed input stream:

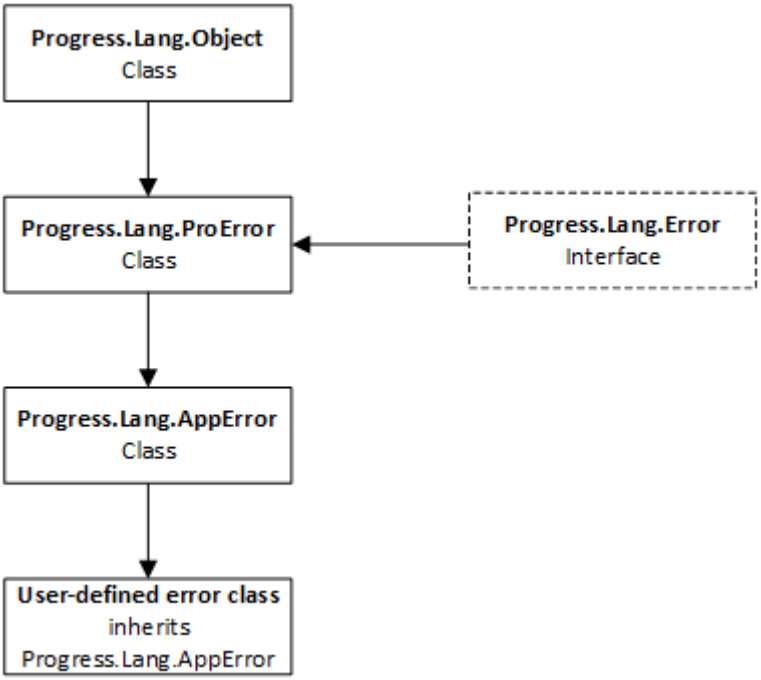
```
DEFINE VARIABLE myParser AS ObjectModelParser NO-UNDO.
DEFINE VARIABLE myConstruct AS JsonConstruct NO-UNDO.
. . .
myParser = NEW ObjectModelParser( ).
myConstruct = myParser:Parse(&WEBSTREAM).
```

Progress.Lang.AppError class

`Progress.Lang.AppError` is the super class of all application errors. An *application error* is simply any collection of data you need to provide necessary information about a condition.

An application error can be raised either by using the `RETURN ERROR` or the `UNDO, THROW` statement.

The following figure represents the hierarchy of ABL classes that provide `Progress.Lang.AppError` with all its features:



`Progress.Lang.ProError` is the super class of all error objects in ABL. From `Progress.Lang.Object` it inherits the basic features of an ABL class. It also implements the `Progress.Lang.Error` interface which provides the basic properties and methods for handling errors.

`Progress.Lang.AppError` adds the properties and methods needed for populating an `AppError` object with your error messages.

You can create your own hierarchy of more complex `AppError` types by creating subclasses of `AppError`.

When the AVM encounters the `RETURN ERROR` statement, it implicitly throws a `Progress.Lang.AppError` object and places any error string in the object's `ReturnValue` property. `ReturnValue` is included in the `AppError` object to provide a bridge between the older functionality of `RETURN ERROR` *`ErrorString`* and error objects. See *ABL Error Handling*, for more information.

Serializable:

Yes

Constructors

The following is the default constructor. This constructor creates an `AppError` object with an empty message list and does not set any properties.

```
PUBLIC AppError ( )
```

The following constructor creates an AppError object and assigns the first message on the object with the values from the *ErrorMessage* and *MessageNumber* arguments. It also sets the NumMessages property to 1. This error message and message number can be accessed with the GetMessage(1) and GetMessageNum(1) methods.

```
PUBLIC AppError ( INPUT ErrorMessage  AS CHARACTER
                  INPUT MessageNumber AS INTEGER )
```

The following constructor creates an AppError object with the ReturnValue property set with the value of the *ReturnValue* parameter. This constructor is used when the AVM implicitly creates an AppError object for a RETURN ERROR *ErrorString* statement. You can also invoke this constructor directly.

```
PUBLIC AppError ( INPUT ReturnValue AS CHARACTER )
```

Note: This constructor does not set an error message. If the AddMessage( ) method is not used to set one, no message is displayed if an object constructed this way is thrown and handled by default error handling.

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

ReturnValue property	—	—

Public Methods

AddMessage( ) method	RemoveMessage( ) method

Public Events

This class does not contain events.

Notes

- All AppError instances, both those created implicitly and user-defined application errors, created with the NEW function (classes) appear on the SESSION:FIRST-OBJECT chain, just like any other user-defined object.

See also

CATCH statement, Progress.Lang.SysError class

Progress.Lang.Class class

Progress.Lang.Class provides type information about an ABL class or an interface. ABL provides a Progress.Lang.Class instance for each ABL class or interface type in the ABL session.

This class is FINAL and cannot be inherited.

Serializable:

No

Constructors

This class contains only a private constructor; you cannot instantiate it directly.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

Package property	SuperClass property	TypeName property

Public Methods

GetClass( ) method (Progress.Lang.Class)	GetConstructor( ) method
GetConstructors( ) method	GetEnumName( ) method
GetEnumNames( ) method	GetEnumValue( ) method
GetEnumValues( ) method	GetEvent( ) method
GetEvents( ) method	GetInterfaces( ) method
GetMethod( ) method	GetMethods( ) method
GetProperties( ) method	GetProperty( ) method
GetPropertyValue( ) method	GetVariable( ) method
GetVariables( ) method	HasStatics( ) method
HasWidgetPool( ) method	Invoke( ) method (Class)
IsA( ) method	IsAbstract( ) method
IsEnum( ) method	IsFinal( ) method
IsFlagsEnum( ) method	IsGeneric( ) method
IsIndexed( ) method	IsInterface( ) method
IsSerializable( ) method	New( ) method
SetPropertyValue( ) method	

Public Events

This class does not contain events.

Notes

- To obtain the object reference for the `Progress.Lang.Class` instance associated with a given ABL class or interface, you can use either the `GetClass( )` method or the GET-CLASS function:
  - You can use the `GetClass( )` method either as an instance method on an ABL class (any `Progress.Lang.Object` instance except a .NET type) or as a static method of `Progress.Lang.Class` using the following syntax:

```
class-reference = object-reference:GetClass ( ).
class-reference = Progress.Lang.Class:GetClass ( expression ).
```

    Where *`object-reference`* is the object reference to an ABL class instance for which you want to get the type information, *`expression`* is a CHARACTER expression that specifies the fully-qualified ABL object type name whose type information you want to get, and *`class-reference`* is the object reference for the `Progress.Lang.Class` instance containing that type information. For example:

```
DEFINE VARIABLE myCustObj AS Progress.Lang.Object NO-UNDO.
DEFINE VARIABLE myType    AS Progress.Lang.Class  NO-UNDO.

ASSIGN
  myType    = Progress.Lang.Class:GetClass("acme.myObjs.CustObj")
  myCustObj = myType:New( ).
```

  - Unlike the `GetClass()` method, GET-CLASS takes a type name as its argument instead of a CHARACTER expression that holds a type name. As a result, USING statements can be applied to the argument, and the compiler checks the type name at compile time. Use GET-CLASS with the following syntax:

```
GET-CLASS( object-type-name )
```

    Where *`object-type-name`* is the type name of a class or interface, using the syntax described in the Type-name syntax reference entry.

- You cannot explicitly delete a `Progress.Lang.Class` instance.
- You can obtain additional information about a class's members using the Get methods listed above (e.g., `GetConstructor( )`, `GetMethods( )`) as well as the properties and methods of the `Progress.Reflect.Constructor`, `Progress.Reflect.Event`, `Progress.Reflect.Method`, `Progress.Reflect.Property`, and `Progress.Reflect.Variable` classes. See the entry of each class for more information.

See also

GET-CLASS function, Progress.Reflect.Constructor class, Progress.Reflect.Event class, Progress.Reflect.Method class, Progress.Reflect.Property class, Progress.Reflect.Variable class

Progress.Lang.Enum class

`Progress.Lang.Enum` provides a common set of methods that all ABL enums inherit.

You cannot inherit from this class.

Serializable:

Yes

Constructors

This class contains only a private constructor; you cannot instantiate it directly.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces (beyond those it may inherit from its base class).

Public Properties

This class does not contain properties.

Public Methods

CompareTo( ) method	Equals( ) method
GetValue( ) method	ToObject( ) method
ToString( ) method (Object)	—

Public Events

This class does not contain events.

Notes

- Although enums are implicitly class based, you cannot instantiate an enum using NEW. You must use the type-name syntax shown in the Enumeration member access entry to access the members defined in an enum or use the GetEnum( ) method to get an enum instance.

Progress.Lang.Error interface

The `Progress.Lang.Error` interface describes a common set of properties and methods that built-in ABL error classes implement. This interface cannot be implemented by a user-defined class. Instead, create a subclass of the `Progress.Lang.AppError` class to create your own type of ABL error object.

This interface is also implemented by an OpenEdge-enhanced version of the .NET System.Exception class (OpenEdge-enhanced). Because of this architecture, ABL recognizes .NET exceptions as error objects and ABL error handling features can recognize, handle, and rethrow .NET exceptions. The interface requires a class to implement the following properties:

Public Properties

CallStack property	NumMessages property	Severity property

The interface requires a class to implement the following methods:

Public Methods

GetMessage( ) method (Class)	GetMessageNum( ) method

Notes

- The `Progress.Lang.Error` interface is the error type that represents all possible errors in ABL, including .NET exceptions and `STOP` conditions with error messages. Specifying this interface in a `CATCH` statement creates an error handler that catches any error. For example:

```
DO ON ERROR UNDO, THROW:
  . . .
  CATCH anyErrorObject AS Progress.Lang.Error:
    . . .
  END CATCH.
END.
```

See also

CATCH statement, Progress.Lang.AppError class, Progress.Lang.ProError class, Progress.Lang.StopError class, Progress.Lang.SysError class

---

Progress.Lang.FlagsEnum class

`Progress.Lang.FlagsEnum` provides a common set of methods that all ABL flag enums inherit.

You cannot inherit from this class.

Serializable:

Yes

Constructors

This class contains only a private constructor; you cannot instantiate it directly.

Super Class

Progress.Lang.Enum class

Interfaces

This class does not implement interfaces (beyond those it may inherit from its base class).

Public Properties

This class does not contain properties.

Public Methods

IsFlagSet( ) method	ToString( ) method (Object)

Public Events

This class does not contain events.

Notes

- There are three additional methods that you can use to set and unset flags for a flag enum type: the ToggleFlag( ) method, the SetFlag( ) method, and the UnsetFlag( ) method. These methods are not part of the implementation of `Progress.Lang.FlagsEnum`, but they are automatically available for all built-in and user-defined flag enums.

Progress.Lang.LockConflict class

The AVM throws an instance of the `Progress.Lang.LockConflict` class and raises the `STOP` condition when a time-out occurs while waiting for a record lock or when the user indicates they want to stop waiting.

You can therefore catch instances of `Progress.Lang.LockConflict` in a `CATCH` block to trap each `STOP` condition raised by a time-out while waiting for a record lock. Its public properties provide information on the context in which the record lock is held.

You cannot inherit from this class.

Serializable:

Yes

Constructors

The class constructors are reserved for system use only.

Super Class

Progress.Lang.Stop class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

Device property	TableName property	User property

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events.

Notes

- To enable older behavior and prevent the AVM from throwing `Progress.Lang.LockConflict` objects, specify the Catch STOP (`-catchStop`) startup parameter with the value `0` (`-catchStop 0`).
- You can specify the time-out period that an ABL application waits for the release of a record lock using the Lock Timeout (`-lkwtmo`) startup parameter.
- In character mode only (running Linux, Unix, or character mode in Windows), the user can also press **CTRL+C** to cancel a wait on a record lock. Note that unlike any other AVM context, pressing this key in the context of waiting for a record lock in character mode both raises the `STOP` condition and throws a `Progress.Lang.LockConflict` object. In any other AVM context, pressing this key throws a `Progress.Lang.UserInterrupt` object instead of a `LockConflict` object. For more information, see the Progress.Lang.UserInterrupt class description.
- Like other class-based objects that are no longer referenced, the AVM automatically deletes `LockConflict` objects using garbage collection.

See also

CATCH statement, ON STOP phrase, Progress.Lang.Stop class

Progress.Lang.Object class

`Progress.Lang.Object` provides a common set of properties and methods that all classes inherit. This set of properties and methods let you write common code to use with any user-defined class.

`Progress.Lang.Object` is the root (ultimate super) class for all ABL classes and for all .NET classes referenced in an ABL session.

Serializable:

Yes

Constructors

```
PUBLIC Object ( )
```

Super Class

This class does not inherit a super class.

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

NEXT-SIBLING property	PREV-SIBLING property	–

Public Methods

Clone( ) method (Progress.Lang.Object)	Equals( ) method
GetClass( ) method (Progress.Lang.Object)	ToString( ) method (Object)

Public Events

This class does not contain events.

Notes

You typically use this class to define a variable or parameter to represent more than one user-defined class type. For example:

- When defining a field in a temp-table as a class, you must specify the class as `Progress.Lang.Object`.
- When defining common methods to use with object instances of different user-defined classes, you must define their parameters using the `Progress.Lang.Object` class.

Progress.Lang.OERequestInfo class

Contains information pertaining to a specific remote procedure call and is transparently passed between application server sessions and clients in order to send and return contextual information. The information is relevant to the execution of a given request and can include client type, client version, client context identifier (CCID), and a deep copy of a client principal object.

OpenEdge provides instances of this object that you can reference using handle attributes of any server object or asynchronous request object created in an application server client session, and using attributes of the SESSION system handle in an application server session (see the Notes for this class).

Serializable:

No

Constructors

This class does not contain a constructor.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

AdapterType property	AgentId property	ClientContextId property

ProcedureName property	RequestId property	SessionId property (OERequestInfo)
ThreadId property	VersionInfo property	–

Public Methods

GetClientPrincipal( ) method (Progress.Lang.OERequestInfo)	SetClientPrincipal( ) method

Public Events

This class does not contain events.

Notes

- In an application server (ABL) client session, you can reference separate instances of this class using the following attributes of a Server object handle or Asynchronous request object handle:
  - REQUEST-INFO attribute
  - RESPONSE-INFO attribute

  The instance is created the first time you access either of these attributes.
- In an application server session, you can reference separate instances of this class using the following attributes of the SESSION system handle:
  - CURRENT-REQUEST-INFO attribute
  - CURRENT-RESPONSE-INFO attribute

  The instance is created the first time you access either of these attributes.
- The AgentId property, the SessionId property (OERequestInfo), and the ThreadId property are for use with PAS for OpenEdge only.

See also

CURRENT-REQUEST-INFO attribute, CURRENT-RESPONSE-INFO attribute, REQUEST-INFO attribute, RESPONSE-INFO attribute

---

Progress.Lang.OEVersionInfo class

Contains information that identifies the specified software component of OpenEdge. OpenEdge provides instances of this class that you can reference using the LOCAL-VERSION-INFO attribute on the SESSION system handle and the VersionInfo property of the Progress.Lang.OERequestInfo class.

Serializable:

No

Constructors

This class does not contain a constructor.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

OEMajorVersion property	OEMinorVersion property	OEMaintVersion property
OEClientType property	–	–

Public Methods

This class does not contain any methods.

Public Events

This class does not contain events.

Notes

The OpenEdge software component that the properties of this class identify depends on the source of the object reference used to obtain this class instance.

See also

LOCAL-VERSION-INFO attribute, Progress.Lang.OERequestInfo class

---

Progress.Lang.ParameterList class

Stores the parameter values for a specific method or constructor call. `Progress.Lang.ParameterList` is a built-in ABL class that provides a common set of properties and methods in support of specific overloaded versions of the `Invoke( )` method and the `New( )` method of the `Progress.Lang.Class` class. A `Progress.Lang.ParameterList` object stores the parameter values for a specified method or constructor call and is used to pass all parameters.

Serializable:

No

Constructor

```
PUBLIC ParameterList ( INPUT number-parameters AS INTEGER )
```

number-parameters

Number of parameters that the object will contain.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

NumParameters property (ParameterList)	–

Public Methods

Clear( ) method (Class)	SetParameter( ) method (Class)

Public Events

This class does not contain events.

Notes

- The `SetParameter( )` method is used to provide data for each parameter element.

See also

Progress.Lang.Class class, Invoke( ) method (Class), New( ) method

Progress.Lang.ProError class

`Progress.Lang.ProError` is the super class for all ABL built-in and user-defined classes that represent errors in the ABL structured error handling model. You cannot directly inherit from this class. Instead, the immediate subclasses of this class represent the two major types of classes in ABL:

- `Progress.Lang.SysError` represents any error generated by the AVM.
- `Progress.Lang.AppError` represents any error your application defines.

`Progress.Lang.ProError` inherits from `Progress.Lang.Object` and therefore inherits all the common methods and properties needed for managing user-defined objects in ABL. It also implements the `Progress.Lang.Error` interface which provides all the properties and methods relevant for an error object.

Specifically, this class provides the functionality to retrieve error messages, error numbers, and the contents of the error call stack.

Serializable:

Yes

Constructors

The class constructors are reserved for system use only.

Super Class

Progress.Lang.Object class

Interfaces

Progress.Lang.Error interface

Public Properties

CallStack property	NumMessages property	Severity property

Public Methods

GetMessage( ) method (Class)	GetMessageNum( ) method

Public Events

This class does not contain events.

Notes

Like any other class-based object that is no longer referenced, the AVM automatically deletes both `Progress.Lang.SysError` objects and `Progress.Lang.AppError` objects using garbage collection.

See also

CATCH statement, Progress.Lang.AppError class, Progress.Lang.Error interface, Progress.Lang.SysError class

Progress.Lang.SoapFaultError class

Represents an ABL Soap fault response message generated by the AVM during a Web Service call from an ABL application.

You can access the object using a `CATCH` statement for the `Progress.Lang.SoapFaultError` object. The `SoapFault` property in this object contains the handle to the Soap-Fault object. The `Soap-Fault-Detail` property of the handle provides the full detail about the Soap fault.

`SoapFaultError` is a type of system error, and therefore you cannot instantiate the object with the NEW function (classes) or create a user-defined class that inherits from it.

Definition

This class is FINAL and cannot be inherited.

Serializable:

Yes

Constructors

The class constructors are reserved for system use only.

Super Class

Progress.Lang.SysError class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

SoapFault property	–	–

Public Methods

This class does not contain methods.

Public Events

This class does not contain events (beyond those it inherits from its base class).

Notes

- `SoapFaultError` objects are put on the `SESSION:FIRST-OBJECT` chain when they are caught by a `CATCH` block.
- Uncaught `SoapFaultError` objects are automatically deleted by the AVM and removed from the `SESSION:FIRST-OBJECT` chain.
- Like other error objects, an error of type `SoapFaultError` can be thrown from an application server and handled by a `CATCH` block on an ABL client. However, the handle-based object that the SoapFault property points to is not recreated during the deserialization of the `SoapFaultError` object. For more information on serialization rules for objects, see the Parameter passing syntax entry.

See also

CATCH statement, Progress.Lang.SysError class

Progress.Lang.Stop class

When the AVM executes the `STOP` statement, it creates an instance of the `Progress.Lang.Stop` class. `Progress.Lang.Stop` is also the immediate super class for the following built-in objects that the AVM throws in response to the associated `STOP` conditions:

- `Progress.Lang.LockConflict` — Thrown when a time-out occurs while waiting for a record lock or when the user indicates they want to stop waiting
- `Progress.Lang.StopAfter` — Thrown when a time-out occurs in response to the `STOP-AFTER` setting on a `DO`, `FOR`, or `REPEAT` block
- `Progress.Lang.UserInterrupt` — Thrown when the user interrupts the AVM by pressing the OS interrupt key

You can therefore catch instances of `Progress.Lang.Stop` and any of its subclasses in a `CATCH` block to trap each associated `STOP` condition.

You cannot inherit from this class.

Serializable:

Yes

Constructors

The class constructors are reserved for system use only.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

CallStack property	–

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events.

Notes

- To enable older behavior and prevent the AVM from throwing `Progress.Lang.Stop` objects, specify the Catch STOP (`-catchStop`) startup parameter with the value `0` (`-catchStop 0`).
- By catching a `Stop` object with a `CATCH` block and testing the object's ultimate derived class type, you can identify what caused the AVM to raise the `STOP` condition and throw the object in the first place.
- The `CallStack` property is always populated for a `STOP` condition, so there is no need to set the `ERROR-STACK-TRACE` attribute of the `SESSION` handle.
- Like any other class-based object that is no longer referenced, the AVM automatically deletes `Stop` objects using garbage collection.

See also

CATCH statement, ON STOP phrase, Progress.Lang.LockConflict class, Progress.Lang.StopAfter class, Progress.Lang.StopError class, Progress.Lang.UserInterrupt class, STOP statement

Progress.Lang.StopAfter class

When enabled, the AVM throws an instance of this `Progress.Lang.StopAfter` class and raises the `STOP` condition when a time-out occurs in response to the `STOP-AFTER` setting on a `DO`, `FOR`, or `REPEAT` block.

You can therefore catch instances of `Progress.Lang.StopAfter` in a `CATCH` block to trap each `STOP` condition raised by a time-out for a block that exceeds its `STOP-AFTER` setting, if any.

You cannot inherit from this class.

Serializable:

Yes

Constructors

The class constructors are reserved for system use only.

Super Class

Progress.Lang.Stop class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

This class does not contain properties (beyond those it inherits from its base class).

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events.

Notes

- To enable older behavior and prevent the AVM from throwing `Progress.Lang.StopAfter` objects, specify the Catch STOP (`-catchStop`) startup parameter with the value `0` (`-catchStop 0`).
- Like any other class-based object that is no longer referenced, the AVM automatically deletes `StopAfter` objects using garbage collection.

See also

CATCH statement, ON STOP phrase, DO statement, FOR statement, Progress.Lang.Stop class, REPEAT statement

---

Progress.Lang.StopError class

When the AVM encounters certain unexpected error conditions, it throws an instance of the `Progress.Lang.StopError` class containing messages associated with the originating error, and raises the `STOP` condition.

`Progress.Lang.StopError` inherits common error handling abilities from `Progress.Lang.ProError`, which implements the `Progress.Lang.Error` interface. You can therefore catch this object in a `CATCH` block to trap error messages associated with the `STOP` condition.

You cannot inherit from this class.

Serializable:

Yes

Constructors

The class constructors are reserved for system use only.

Super Class

Progress.Lang.SysError class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

This class does not contain properties (beyond those it inherits from its base class).

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events.

Notes

- To enable older behavior and prevent the AVM from throwing `Progress.Lang.StopError` objects, specify the Catch STOP (`-catchStop`) startup parameter with the value `0` (`-catchStop 0`).
- `StopError` objects are put on the `SESSION:FIRST-OBJECT` chain when they are caught by a `CATCH` block.
- Uncaught `StopError` objects are automatically deleted by the AVM and removed from the `SESSION:FIRST-OBJECT` chain.

See also

CATCH statement, ON STOP phrase, Progress.Lang.Error interface, Progress.Lang.ProError class , Progress.Lang.Stop class

---

Progress.Lang.SysError class

When an ABL statement raises the error condition, the AVM throws an error. These errors are represented by the `Progress.Lang.SysError` class. `Progress.Lang.SysError` inherits common error handling abilities from `Progress.Lang.ProError`.

You cannot inherit from this class.

Serializable:

Yes

Constructors

The class constructors are reserved for system use only.

Super Class

Progress.Lang.ProError class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

This class does not contain properties (beyond those it inherits from its base class).

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events.

Notes

- `SysError` objects are put on the SESSION:FIRST-OBJECT chain when they are caught by a CATCH block.
- Uncaught `SysError` objects are automatically deleted by the AVM and removed from the SESSION:FIRST-OBJECT chain.

See also

CATCH statement, Progress.Lang.AppError class, Progress.Lang.Error interface, Progress.Lang.ProError class, Progress.Lang.SoapFaultError class, Progress.Lang.StopError class

---

Progress.Lang.UserInterrupt class

The AVM throws an instance of the `Progress.Lang.UserInterrupt` class and raises the `STOP` condition when the AVM detects that the user has pressed the **STOP** key (usually mapped to **CTRL+C** on Unix/Linux or **CTRL+Break** in Windows).

You can therefore catch instances of `Progress.Lang.UserInterrupt` in a `CATCH` block to trap each `STOP` condition raised when the user interrupts the AVM in this way.

You cannot inherit from this class.

Serializable:

Yes

Constructors

The class constructors are reserved for system use only.

Super Class

Progress.Lang.Stop class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

This class does not contain properties (beyond those it inherits from its base class).

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events.

Notes

- To enable older behavior and prevent the AVM from throwing `Progress.Lang.UserInterrupt` objects, specify the Catch STOP (`-catchStop`) startup parameter with the value `0` (`-catchStop 0`).
- In character mode only (running Linux, Unix, or character mode in Windows), when in the context of waiting for a record lock, pressing **CTRL+C** cancels the wait and causes the AVM to raise the STOP condition and throw a `Progress.Lang.LockConflict` object (instead of throwing a `Progress.Lang.UserInterrupt` object as in other AVM contexts). For more information, see Progress.Lang.LockConflict class.
- Like other class-based objects that are no longer referenced, the AVM automatically deletes `UserInterrupt` objects using garbage collection.

See also

CATCH statement, ON STOP phrase, Progress.Lang.Stop class

Progress.Reflect.AccessMode enumeration

This enumeration describes the access mode of a given class member. Its members are returned as values of the `AccessMode` property of `Progress.Reflect.Constructor`, `Progress.Reflect.Event`, `Progress.Reflect.Method`, `Progress.Reflect.Property`, and `Progress.Reflect.Variable` instances.

This class is FINAL and cannot be inherited.

Serializable:

Yes

Super Class

Progress.Lang.Enum class

Members

Member	Description
`Public`	The class member is public.
`PackageProtected`	The class member is package-protected.
`Protected`	The class member is protected.

Member	Description
`PackagePrivate`	The class member is package-private.
`Private`	The class member is private.

Public Methods

GetEnum( ) method	–

Progress.Reflect.Constructor class

`Progress.Reflect.Constructor` provides information about an ABL or .NET class constructor.

This class is FINAL and cannot be inherited.

Serializable:

No

Constructors

This class only contains a private constructor.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces.

Public Properties

AccessMode property	Name property (Reflect)	NumParameters property (Reflect)
OriginatingClass property	–	–

Public Methods

GetParameters( ) method	Invoke( ) method (Constructor)
ToString( ) method (Reflect)	–

Progress.Reflect.DataType enumeration

This enumeration describes the data type of a given class member. Its members are returned for the `ReturnType` property of `Progress.Reflect.Method` instances and for the `DataType` property of `Progress.Reflect.Variable`, `Progress.Reflect.Parameter`, and `Progress.Reflect.Property` instances.

This class is FINAL and cannot be inherited.

Serializable:

Yes

Super Class

Progress.Lang.Enum class

Members

`AnyType`[87]	`Buffer`	`Byte`[88]
`Character`	`COMHandle`	`DataSet`
`DataSetHandle`	`Date`	`DateTime`
`DateTimeTZ`	`Decimal`	`Double`[88]
`Float`[88]	`Handle`	`Int64`
`Integer`	`Logical`	`Longchar`
`Memptr`	`Object`	`Raw`
`Recid`	`Rowid`	`Short`[88]
`SingleCharacter`[88]	`Table`	`TableHandle`
`UnsignedByte`[88]	`UnsignedInt`[88]	`UnsignedInt64`[88]
`UnsignedShort`[88]	`Void`[89]	

Public Methods

GetEnum( ) method	–

[87] Indicates that any data type could be returned or passed as a parameter, including an array of any data type.
[88] Applies only to .NET objects.
[89] Indicates a VOID method. This member is only returned for a method object.

Progress.Reflect.Event class

`Progress.Reflect.Event` provides information about an ABL or .NET class event.

This class is FINAL and cannot be inherited.

Serializable:

No

Constructors

This class only contains a private constructor.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces.

Public Properties

AccessMode property	DeclaringClass property	IsAbstract property
IsOverride property	IsStatic property	Name property (Reflect)
NumHandlerParameters property	OriginatingClass property	–

Public Methods

GetHandlerParameters( ) method	ToString( ) method (Reflect)	–

Progress.Reflect.Flags enumeration

This enumeration describes the type of class members you want as results from `GetConstructor()`, `GetConstructors()`, `GetEvent()`, `GetEvents()`, `GetMethod()`, `GetMethods()`, `GetProperty()`, `GetProperties()`, `GetVariable()`, or `GetVariables()`. Because `Progress.Reflect.Flags` is a flag enumeration, multiple flags can be set at one time.

This class is FINAL and cannot be inherited.

Serializable:

Yes

Super Class

Progress.Lang.FlagsEnum class

Members

Member	Description
`Public`	Returns public class members.
`PackageProtected`	Returns package-protected class members.
`Protected`	Returns protected class members.
`PackagePrivate`	Returns package-private class members.
`Private`	Returns private class members.
`Static`	Returns static class members.
`Instance`	Returns instance class members.
`DeclaredOnly`	Returns class members only declared on the given class type, excluding inherited members.

Public Methods

GetEnum( ) method	SetFlag( ) method
ToggleFlag( ) method	UnsetFlag( ) method

Notes

- The members for `Progress.Reflect.Flags` can be thought of as three groups:
  - Access mode flags: `Public`, `PackageProtected`, `Protected`, `PackagePrivate`, `Private`
  - Scope flags: `Static`, `Instance`
  - Class-level flags: `DeclaredOnly`

  No results will be returned unless you're using at least one flag from the access mode set and one flag from the scope set are indicated. (The exception to this is the `GetConstructor()` method, which will raise an error if the `Static` flag is set. See the GetConstructor( ) method entry for details.)
- You can specify more than one flag from the access mode and scope groups. For example, setting both `Public` and `Private` flags for use with `GetMethods()` will return all methods that are either public or private as long as they satisfy any other criteria you indicate.
- By default, a method used with a `Progress.Reflect.Flags` parameter will return class members from the entire class hierarchy unless `DeclaredOnly` is specified. (The exception to this is the `GetConstructor()` method, which will not return a super class's constructor. See the GetConstructor( ) method entry for details.)
- Use bitwise OR to set multiple flags. This code fragment will set a `Progress.Reflect.Flags` instance to be used to indicate public, static class members:

```
DEFINE VARIABLE myFlags AS Progress.Reflect.Flags.

myFlags = Progress.Reflect.Flags:Public OR Progress.Reflect.Flags:Static.
```

Alternatively, use `SetFlag( )` to set multiple flags. The following code achieves the same result as the previous example:

```
DEFINE VARIABLE myFlags AS Progress.Reflect.Flags.

myFlags = Progress.Reflect.Flags:Public.
myFlags = myFlags:SetFlag(Progress.Reflect.Flags:Static).
```

Progress.Reflect.Method class

`Progress.Reflect.Method` provides information about an ABL or .NET class method.

This class is FINAL and cannot be inherited.

Serializable

No

Constructors

This class only contains a private constructor.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces.

Public Properties

AccessMode property	DeclaringClass property	IsAbstract property
IsFinal property	IsOverride property	IsStatic property
Name property (Reflect)	NumParameters property (Reflect)	OriginatingClass property
ReturnExtent property	ReturnType property	ReturnTypeName property

Public Methods

GetParameters( ) method	Invoke( ) method (Method)
ToString( ) method (Reflect)	–

Progress.Reflect.Parameter class

`Progress.Reflect.Parameter` provides parameter information for an ABL or .NET class constructor, event handler, or method.

This class is FINAL and cannot be inherited.

Serializable

No

Constructors

This class only contains a private constructor.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces.

Public Properties

DataType property	DataTypeName property (Reflect)	Extent property
Mode property	Name property (Reflect)	Position property (Parameter)

Public Methods

ToString( ) method (Reflect)	–

Progress.Reflect.ParameterMode enumeration

This enumeration describe the mode of a given parameter. Its members are returned as values of the `Mode` property of `Progress.Reflect.Parameter` instances. Because `Progress.Reflect.ParameterMode` is a flag enumeration, multiple flags can be set at one time.

This class is FINAL and cannot be inherited.

Serializable:

Yes

Super Class

Progress.Lang.FlagsEnum class

Members

Member	Description
`Input`	The parameter is INPUT.
`Output`	The parameter is OUTPUT.
`InputOutput`	The parameter is INPUT-OUTPUT. This member is a combination of the `Input` and `Output` members.
`Append`	APPEND is specified on the parameter definition (for a temp table or ProDataSet).
`Bind`	BIND is specified on the parameter definition (for a temp table or ProDataSet).

Public Methods

GetEnum( ) method	SetFlag( ) method
ToggleFlag( ) method	UnsetFlag( ) method

Progress.Reflect.Property class

`Progress.Lang.Property` provides information about an ABL or .NET class property.

This class is FINAL and cannot be inherited.

Serializable:

No

Constructors

This class only contains a private constructor.

Super Class

Progress.Reflect.Variable class

Interfaces

This class does not implement interfaces.

Public Properties

Progress.Reflect.Variable class

`Progress.Lang.Variable` provides information about an ABL or .NET class data member or property.

This class is FINAL and cannot be inherited.

Serializable

No

Constructors

This class only contains a private constructor.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces.

Public Properties

AccessMode property	DataType property	DataTypeName property (Reflect)
DeclaringClass property	Extent property	IsStatic property
Name property (Reflect)	OriginatingClass property	–

Public Methods

Get( ) method	Set( ) method (Reflect)
ToString( ) method (Reflect)	–

Progress.Security.PAMStatus class

A class whose static members are the valid status codes that can be returned by an ABL authentication callback procedure, which can be called during the execution of an OpenEdge user authentication operation. See *OpenEdge Programming Interfaces* for more information about authentication callback procedures.

Definition

This class is FINAL and cannot be inherited.

Serializable:

No

Constructors

This class only contains a private constructor.

Super Class

Progress.Lang.Object class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

AccessDenied property	AccessRevoked property	AccountDisabled property
AccountExpired property	AuthenticationAccess property	AuthenticationFailed property
Custom property	InvalidConfiguration property	LoginDenied property
LoginLockout property	MaxTries property	MissingCredentials property
PasswordExpired property	PermissionDenied property	Success property
UnknownUser property	–	–

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events.

Notes

- An authentication callback procedure must return one of the values specified on the above table, otherwise the AVM raises a standard authentication failure error (`PAMStatus:AuthenticationFailed`).
- The values associated with each property of this class conform to Identity Management (IdM) standards. The AVM maps them to the corresponding values defined by the RFC 86 PAM (Pluggable Authentication Modules) API specification.

Progress.Util.DateTimeHelper class
(Windows only; GUI for .NET only)

A helper class that provides a set of static `Create( )` method overloads for initializing DATETIME values in ABL.

Note: This class is used primarily by the Visual Designer during code generation. You should not expect to use this class directly.

Serializable:

No

Constructors

This class does not contain a constructor.

Super Class

This class does not inherit a super class.

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

This class does not contain properties (beyond those it inherits from its base class).

Public Methods

Create( ) method (DATETIME)	–

Public Events

This class does not contain events (beyond those it inherits from its base class).

See also

Progress.Util.DecimalHelper class, Progress.Util.StringHelper class

---

Progress.Util.DecimalHelper class
(Windows only; GUI for .NET only)

A helper class that provides a set of static `Create( )` method overloads for initializing DECIMAL values in ABL.

Note: This class is used primarily by the Visual Designer during code generation. You should not expect to use this class directly.

Serializable:

No

Constructors

This class does not contain a constructor.

Super Class

This class does not inherit a super class.

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

This class does not contain properties (beyond those it inherits from its base class).

Public Methods

Create( ) method (DECIMAL)	–

Public Events

This class does not contain events (beyond those it inherits from its base class).

See also

Progress.Util.DateTimeHelper class, Progress.Util.StringHelper class

---

Progress.Util.EnumHelper class
(Windows only; GUI for .NET only)

A helper class that provides a set of static methods you can use to perform operations on .NET enumeration types in ABL.

Serializable:

No

Constructors

This class does not contain a constructor.

Super Class

This class does not inherit a super class.

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

This class does not contain properties.

Public Methods

Add( ) method (Progress.Util.EnumHelper)	And( ) method
AreEqual( ) method	AreNotEqual( ) method
Complement( ) method	IsGreater( ) method
IsGreaterOrEqual( ) method	IsLess( ) method
IsLessOrEqual( ) method	Or( ) method
Subtract( ) method	Xor( ) method

Public Events

This class does not contain events (beyond those it inherits from its base class).

Notes

- The functionality for all of the methods of this class, except for `Add( )` and `Subtract( )`, can be accomplished for .NET enums using ABL operators (e.g., you can use GE or >= operator instead of `IsGreaterOrEqual( )`, NOT operator (bitwise) instead of `Complement( )`, etc.). These methods are still supported for backward compatibility.
- Some `Progress.Util.EnumHelper` methods return a `System.Enum` class instance. If you want to use the returned `System.Enum` instance as a different enumeration type, you must cast the instance to that specific type using the CAST function. For example:

```
USING System.Windows.Forms.* FROM ASSEMBLY.
USING Progress.Util.* FROM ASSEMBLY.

DEFINE VARIABLE myButton AS Button.
DEFINE VARIABLE myStyle  AS System.Enum.

myButton = NEW Button().
myStyle  = EnumHelper:Or( AnchorStyles:Bottom, AnchorStyles:Right ).
myButton:Anchor = CAST( myStyle, AnchorStyles ).

/* Displays "Bottom, Right" */
DISPLAY STRING( myButton:Anchor ) FORMAT "x(20)".
```

- For information on the `System.Enum` class and .NET enumeration types, refer to the .NET Framework class library.

See also

CAST function, DELETE OBJECT statement

Progress.Util.ResourceHelper class
(Windows only; GUI for .NET only)

A helper class that provides a static method you can use to retrieve the contents of a .NET XML resource (`.resx`) file associated with a given .NET form class.

Serializable:

No

Constructors

This class does not contain a constructor.

Super Class

This class does not inherit a super class.

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

This class does not contain properties (beyond those it inherits from its base class).

Public Methods

Load( ) method (Class)	–

Public Events

This class does not contain events (beyond those it inherits from its base class).

Notes

For more information on using this `ResourceHelper` class, see the topics on creating and using forms and controls in *Use GUI for .NET Objects in ABL Applications*.

See also

Progress.Windows.Form class

Progress.Util.StringHelper class
(Windows only; GUI for .NET only)

A helper class that provides a set of static `Create( )` method overloads for initializing CHARACTER values in ABL.

Note: This class is used primarily by the Visual Designer during code generation. You should not expect to use this class directly.

Serializable:

No

Constructors

This class does not contain a constructor.

Super Class

This class does not inherit a super class.

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

This class does not contain properties (beyond those it inherits from its base class).

Public Methods

Create( ) method (CHARACTER)	–

Public Events

This class does not contain events (beyond those it inherits from its base class).

See also

Progress.Util.DateTimeHelper class, Progress.Util.DecimalHelper class

---

Progress.Util.TypeHelper class
(Windows only; GUI for .NET only)

A helper class that provides a static method you can use to get a `System.Type` instance for a .NET class using a fully-qualified class name. Use this helper class when you need to get the type of an object for which you have no instance and cannot instantiate an instance.

Serializable:

No

Constructors

This class does not contain a constructor.

Super Class

This class does not inherit a super class.

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

This class does not contain properties (beyond those it inherits from its base class).

Public Methods

GetType( ) method (TypeHelper)	–

Public Events

This class does not contain events (beyond those it inherits from its base class).

Note

If the class has not been previously referenced, the assembly containing the class must be listed in the `assemblies.xml` file, which needs to be on the PROPATH at run time.

Progress.Web.IWebHandler interface

The `Progress.Web.IWebHandler` interface provides common functionality for classes designed to handle HTTP requests received by WebSpeed applications running on a Progress Application Server. This interface is implemented by `OpenEdge.Web.WebHandler`, but if you create your own handler class that does not extend `OpenEdge.Web.WebHandler`, your class must also implement this interface.

Classes that implement this interface are registered in the `openedge.properties` file of the application.

Public Properties

The interface has no public properties.

Public Methods

HandleRequest( ) method (IWebHandler)	–

Progress.Windows.Form class
(Windows only; GUI for .NET only)

Lets you create the following types of .NET forms in an ABL session (that co-exist with traditional ABL windows and dialog boxes):

- A modal form, which is similar to an ABL modal dialog box.
- A non-modal form, which is similar to an ABL non-modal window.
- A non-modal Multiple-Document Interface (MDI) parent form, which is not supported by the traditional ABL GUI.

Serializable:

No

Constructors

```
PUBLIC Form ( )
```

Super Class

`System.Windows.Forms.Form` class (from the .NET Framework)

Interfaces

Progress.Windows.IForm interface

Public Properties

DisposeDialogOnClose property	NextForm property	PrevForm property
ProWinHandle property		

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events (beyond those it inherits from its base class).

Example

The following example shows a simple ABL class that inherits from `Progress.Windows.Form` create a non-modal form with two buttons that looks behaves like a dialog box, except that it is non-modal:

```
USING System.Windows.Forms.* FROM ASSEMBLY.

CLASS DemoForm INHERITS Progress.Windows.Form:


  /* Variables for buttons on the form */
  DEFINE PRIVATE VARIABLE OkBtn     AS Button.
  DEFINE PRIVATE VARIABLE CancelBtn AS Button.

  CONSTRUCTOR DemoForm ( ):
    InitializeComponent( ).
  END CONSTRUCTOR.

  /* Event handlers for buttons on the form */
  METHOD PUBLIC VOID okButton_Click
    ( sender AS System.Object, e AS System.EventArgs ):
    THIS-OBJECT:DialogResult = DialogResult:Ok.
    THIS-OBJECT:Close ( ).
  END METHOD.

  METHOD PRIVATE VOID cancelButton_Click
    ( sender AS System.Object, e AS System.EventArgs ):
    THIS-OBJECT:DialogResult = DialogResult:Cancel.
    THIS-OBJECT:Close ( ).
  END METHOD.

  /* Display and wait for this non-modal form to close.
     The caller blocks at a call to this method         */
  METHOD PUBLIC VOID Wait ( ):
    WAIT-FOR Application:Run ( THIS-OBJECT ).
  END METHOD.

  METHOD PRIVATE VOID InitializeComponent ( ):
    /* Instantiate button classes */
    OkBtn     = NEW Button ( ).
    CancelBtn = NEW Button ( ).

    /* Initialize the form and buttons */
    THIS-OBJECT:Text       = "<form title>".
    OkBtn:Text             = "OK".
    CancelBtn:Text         = "Cancel".
    OkBtn:DialogResult     = DialogResult:OK.
    CancelBtn:DialogResult = DialogResult:Cancel.

    /* Set the size and location of the form and buttons */
    THIS-OBJECT:Size   = NEW System.Drawing.Size( 300, 300 ).
    OkBtn:Size         = NEW System.Drawing.Size( 75, 23 ).
    OkBtn:Location     = NEW System.Drawing.Point( 121, 231 ).
    OkBtn:Anchor       = CAST( Progress.Util.EnumHelper:Or
      (AnchorStyles:Bottom,
       AnchorStyles:Right ),
       AnchorStyles).
    CancelBtn:Size     = NEW System.Drawing.Size( 75, 23 ).
    CancelBtn:Location = NEW System.Drawing.Point( 205, 231 ).
```

```
    CancelBtn:Anchor   = CAST( Progress.Util.EnumHelper:Or
      (AnchorStyles:Bottom,
       AnchorStyles:Right ),
       AnchorStyles ).

    /* Subscribe to events */
    OkBtn:Click:Subscribe( okButton_Click ).
    CancelBtn:Click:Subscribe( cancelButton_Click ).

    /* Add buttons to the form */
    THIS-OBJECT:Controls:Add( OkBtn ).
    THIS-OBJECT:Controls:Add( CancelBtn ).
  END METHOD.

END CLASS.
```

To instantiate `DemoForm` and display the non-modal form, you can run a procedure that contains the following code:

```
DEFINE VARIABLE DisplayFormDemo AS CLASS DemoForm.

DisplayFormDemo = NEW DemoForm( ).
DisplayFormDemo:Wait( ).
```

For more examples, see the topics on creating and using forms and controls in *Use GUI for .NET Objects in ABL Applications*.

Notes

- A modal form, which is similar to an ABL dialog box, has the following characteristics:
  - The form is displayed using the .NET `ShowDialog( )` method invoked from a WAIT-FOR statement.
  - The form must be closed before you can access any other non-modal forms.
  - The form does not appear in the system taskbar.
  - The form typically has a dialog box border style.
  - The form typically has one or more buttons to close the form (such as **OK** or **Cancel**).
  - The form has no affordances in the title bar for minimizing or maximizing the form.
  - The form cannot contain a menu bar.
- A non-modal form, which is similar to an ABL window, has the following characteristics:
  - The form is displayed using either the .NET `Show( )` method or the `Application:Run( )` method invoked from a WAIT-FOR statement.
  - The form does not prevent access to other forms.
  - The form does appear in the system taskbar.
  - The form typically has a window border style.
  - The form has affordances in the title bar for minimizing, maximizing, and closing the form.
  - The form typically contains a menu bar.
- A Multiple-Document Interface (MDI) parent form, which is the foundation for an MDI application, has the following characteristics:

- The form is a non-modal container form.
- The form is displayed using the .NET `Application:Run( )` method invoked from a WAIT-FOR statement.
- The form does not prevent access to other forms.
- The form contains a main menu bar, toolbar, and status bar.
- The form has affordances in the title bar for minimizing, maximizing, and closing the form.
- The form does appear in the system taskbar.
- The form has one or more associated child forms that provide for user interaction with the application. Child forms can be minimized, maximized, and resized within the MDI container.

  Note: To associate a child form with a parent form, use the `Progress.Windows.Form` class to create a child form and set the child's .NET `MDIParent` property to the parent form.

- The form contains a client area in which all child forms are displayed.
- When a parent form is closed, the `FormClosing` events of all child forms are raised before the parent form's `FormClosing` event is raised.

- To display a modal form, you must invoke the WAIT-FOR statement with the form's input-blocking `ShowDialog( )` method, which does the following:
  - Suspends execution of the current block.
  - Prepares and displays the specified form in a wait (blocking) state until the .NET form is closed by either the user or the application.

  Execution resumes after the specified form is closed.

  Note: Unlike in a .NET application, invoking the `ShowDialog( )` method in an ABL session does not, by itself, cause modal forms to display. Instead, you must invoke the WAIT-FOR statement to display these forms. For more information, see the WAIT-FOR statement (.NET objects) reference entry in this book.

  For information on the .NET `ShowDialog( )` method inherited from the `System.Windows.Forms.Form` class, refer to the .NET Framework class library.

- To display a non-modal form, you must invoke the WAIT-FOR statement with the input-blocking `Application:Run( )` method, which does the following:
  - Suspends execution of the current block
  - Prepares the specified form by invoking its .NET `Show( )` method
  - Displays the specified form in a wait (blocking) state until the form is closed by either the user or the application

  This statement will also display any additional non-modal forms that you have previously prepared by setting their `Visible` property to TRUE and invoking their .NET `Show( )` method. After the specified form is closed, any additional non-modal forms currently displayed are also closed and execution resumes.

  If you do not specify a form, the WAIT-FOR statement displays and blocks for input on any non-modal forms that have been previously prepared by setting their `Visible` property to TRUE and invoking their .NET `Show( )` method. In this case, all non-modal forms currently displayed can be closed and execution can resume only after the application invokes the `Application:Exit( )` method.

Note: Unlike in a .NET application, invoking the `Show( )` method in an ABL session does not, by itself, cause non-modal forms to display. Instead, you must invoke the WAIT-FOR statement to display these forms. For more information, see the WAIT-FOR statement (.NET objects) reference entry in this book.

For information on the .NET `Show( )` and `Application:Run( )` methods inherited from the `System.Windows.Forms.Form` class, refer to the .NET Framework class library.

- To handle a .NET form event while the WAIT-FOR statement is blocking execution, you must write and subscribe an event handler to the specified event prior to invoking the WAIT-FOR statement. For more information on event handlers for .NET events, see the Class Events Reference.
- In most cases, it is best to create a delegate class that encapsulates the form definition and related methods. For information on using delegate classes for forms, see *Use GUI for .NET Objects in ABL Applications*.
- For information on the public class members inherited from the `System.Windows.Forms.Form` class, refer to the .NET Framework class library.

See also

Progress.Util.ResourceHelper class, WAIT-FOR statement (.NET and ABL)

Progress.Windows.FormProxy class
(Windows only; GUI for .NET only)

A .NET form wrapper for an ABL window. ABL provides a `Progress.Windows.FormProxy` instance for each ABL window that you create in an ABL session where a .NET form is referenced.

Note: When mixing ABL windows and .NET forms in a single application, you can access all windows and forms on a single form chain using the same form-based properties and attributes.

Serializable:

No

Constructors

This class contains a protected constructor; you cannot instantiate it directly.

Super Class

`System.Object` class (from the .NET Framework)

Interfaces

Progress.Windows.IForm interface

Public Properties

NextForm property	PrevForm property	ProWinHandle property

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events (beyond those it inherits from its base class).

Notes

- This class allows you to manage ABL windows along with .NET forms in an ABL session. It allows ABL windows to participate in the session form chain (starting with SESSION:FIRST-FORM and ending with SESSION:LAST-FORM) and to be referenced by form-oriented ABL elements such as the ACTIVE-FORM system reference. However, all program interaction with ABL windows must still be accomplished using ABL window attributes, methods, and events on the window handle (specified by the `ProWinHandle` property). For information on using ABL windows along with .NET forms in your application, see *Use GUI for .NET Objects in ABL Applications*.
- ABL does not create an instance of this class for the ABL default window, as the default window is typically not used in a GUI application.
- ABL handles all creation and destruction for instances of this class. Therefore, instances of this class never appear on the session object chain, and their NEXT-SIBLING property and PREV-SIBLING property (inherited from the Progress.Lang.Object class) are always set to the Unknown value (?).
- For information on the public class members inherited from the `System.Object` class, refer to the .NET Framework class library.

See also

ACTIVE-FORM system reference, FIRST-FORM attribute, LAST-FORM attribute

---

Progress.Windows.IForm interface
(Windows only; GUI for .NET only)

Provides for interoperability between ABL GUI windows and .NET forms. All OpenEdge .NET form-based classes in ABL implement this interface.

Public Properties

NextForm property	PrevForm property	ProWinHandle property

Public Methods

This interface does not contain method prototypes.

Notes

- There is no need for user-defined classes to implement this interface.
- For information on the use of this interface when working with the form chain in an OpenEdge session, see *Use GUI for .NET Objects in ABL Applications*.

See also

Progress.Windows.Form class, Progress.Windows.FormProxy class

Progress.Windows.MDIChildForm class
(Windows only; GUI for .NET only)

A .NET form designed for use as an MDI child form that allows you to embed the client area of an ABL window for display in the client area of the form. This allows the widgets in the ABL client area to be displayed in the .NET MDI child form, but also allows you to interact with these widgets much as if they were still displayed in the original ABL window.

Serializable:

No

Constructor

```
PUBLIC MDIChildForm ( INPUT mdi-parent AS System.Windows.Forms.Form,
                      INPUT abl-window AS HANDLE                    )
```

mdi-parent

An object reference to a .NET form that you have initialized as an MDI parent form.

abl-window

A handle to an ABL window whose client area you want to embed in the MDI child form.

Super Class

Progress.Windows.Form class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

EmbeddedWindow property	–	–

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events (beyond those it inherits from its base class).

Notes

- To embed the client area of an ABL window in a .NET MDI child form, instantiate this class to create the form, setting the constructor parameters to reference the parent MDI form and the ABL window whose client area you want to embed. If you later want to change the ABL window whose client area is embedded in this form, you can set its `EmbeddedWindow` property to the handle of a different ABL window. Note that you can make this change only if the previous ABL window has not been realized, or after you first delete this realized window before resetting the property. For more information on how ABL supports the behavior of

the ABL window and its client area embedded in a `Progress.Windows.MdiChildForm`, see the EmbeddedWindow property reference entry.

- The client area of a `Progress.Windows.MdiChildForm` is designed to contain only the ABL widgets from the client area of a single ABL window. If you want to create an MDI child form that contains the client areas of multiple ABL windows, initialize a Progress.Windows.Form object as an MDI child form and add a `Progress.Windows.WindowContainer` object to this form for each ABL window.

Caution: You can also add .NET controls to the client area of a `Progress.Windows.MdiChildForm` by using the `Add( )` method on the inherited `System.Windows.Forms.Control+ControlCollection` inner class referenced by the inherited `Controls` property. However, Progress Software Corporation recommends that you do not use this method to add .NET controls to a `Progress.Windows.MdiChildForm` because this can result in unexpected control and form behavior.

- The menus from an embedded ABL window are not added either to the MDI child or its parent forms. You must add any associated menu and toolbar functionality to the appropriate form using corresponding .NET controls.
- The message and status areas of an embedded ABL window are not included in the MDI child form. Any messages normally displayed in the message area of the ABL window appear in an ABL alert box and any status messages normally displayed in the status area of the ABL window are ignored.
- As you resize the form, either manually or programmatically, the AVM automatically resizes the embedded client area of the ABL window to fill the client area of the `Progress.Windows.MdiChildForm`.
- The AVM passes all unhandled keystrokes in an embedded ABL client area to the form that contains it. This means that you can interact with menus and use menu and toolbar accelerator keys even when focus is on the embedded client area.

See also

Progress.Windows.WindowContainer class

Progress.Windows.MDIChildRadForm class
(Windows only; GUI for .NET only)

`Progress.Windows.MDIChildRadForm` is a .NET form based on the `RadForm` class available in Telerik UI for `WinForms`. It has the same functionality as described in the entry for the Progress.Windows.MDIChildForm class.

Serializable:

No

Constructor

```
PUBLIC MDIChildRadForm ( INPUT mdi-parent AS System.Windows.Forms.Form,
                         INPUT abl-window AS HANDLE )
```

mdi-parent

An object reference to a .NET form that you have initialized as an MDI parent form.

abl-window

A handle to an ABL window whose client area you want to embed in the MDI child form.

Super Class

`Telerik.WinControls.UI.RadForm` class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

See also

Progress.Windows.MDIChildForm class

Progress.Windows.MDIChildRadRibbonForm class
(Windows only; GUI for .NET only)

`Progress.Windows.MDIChildRadRibbonForm` is a .NET form based on the `RadRibbonForm` class available in Telerik UI for `WinForms`. It has the same functionality as described in the entry for the Progress.Windows.MDIChildForm class.

Serializable:

No

Constructor

```
PUBLIC MDIChildRadRibbonForm ( INPUT mdi-parent AS System.Windows.Forms.Form,
                               INPUT abl-window AS HANDLE )
```

mdi-parent

An object reference to a .NET form that you have initialized as an MDI parent form.

abl-window

A handle to an ABL window whose client area you want to embed in the MDI child form.

Super Class

`Telerik.WinControls.UI.RadRibbonForm` class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

See also

Progress.Windows.MDIChildForm class

Progress.Windows.MDIChildShapedForm class
(Windows only; GUI for .NET only)

`Progress.Windows.MDIChildShapedForm` is a .NET form based on the `ShapedForm` class available in Telerik UI for `WinForms`. It has the same functionality as described in the entry for the Progress.Windows.MDIChildForm class.

Serializable:

No

Constructor

```
PUBLIC MDIChildShapedForm ( INPUT mdi-parent AS System.Windows.Forms.Form,
                            INPUT abl-window AS HANDLE )
```

mdi-parent

An object reference to a .NET form that you have initialized as an MDI parent form.

abl-window

A handle to an ABL window whose client area you want to embed in the MDI child form.

Super Class

`Telerik.WinControls.UI.ShapedForm` class

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

See also

Progress.Windows.MDIChildForm class

Progress.Windows.OERadForm class
(Windows only; GUI for .NET only)

`Progress.Windows.OERadForm` is a .NET form based on the `RadForm` class available in Telerik UI for `WinForms`. It has the same functionality as described in the entry for the Progress.Windows.Form class.

Serializable:

No

Constructors

```
PUBLIC OERadForm ( )
```

Super Class

`Telerik.WinControls.UI.RadForm` class

Interfaces

Progress.Windows.IForm interface

See also

Progress.Windows.Form class

Progress.Windows.OERadRibbonForm class
(Windows only; GUI for .NET only)

`Progress.Windows.OERadRibbonForm` is a .NET form based on the `RadRibbonForm` class available in Telerik UI for `WinForms`. It has the same functionality as described in the entry for the Progress.Windows.Form class.

Serializable:

No

Constructors

```
PUBLIC OERadRibbonForm ( )
```

Super Class

`Telerik.WinControls.UI.RadRibbonForm` class

Interfaces

Progress.Windows.IForm interface

See also

Progress.Windows.Form class

Progress.Windows.OEShapedForm class
(Windows only; GUI for .NET only)

`Progress.Windows.OEShapedForm` is a .NET form based on the `ShapedForm` class available in Telerik UI for `WinForms`. It has the same functionality as described in the entry for the Progress.Windows.Form class.

Serializable:

No

Constructors

```
PUBLIC OEShapedForm ( )
```

Super Class

`Telerik.WinControls.UI.ShapedForm` class

Interfaces

Progress.Windows.IForm interface

See also

Progress.Windows.Form class

Progress.Windows.UserControl class
(Windows only; GUI for .NET only)

A control container class that lets you combine and use a set of user-defined controls as a single control in ABL.

Serializable:

No

Constructors

```
PUBLIC UserControl ( )
```

Super Class

`System.Windows.Forms.UserControl` class (from the .NET Framework)

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

This class does not contain properties (beyond those it inherits from its base class).

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events (beyond those it inherits from its base class).

Example

The following example shows a simple class that creates a user control container that contains a text box and a button:

```
USING System.Windows.Forms.* FROM ASSEMBLY.

CLASS DemoUserControl INHERITS Progress.Windows.UserControl:

 /* Variable for text box and button in control container */
  DEFINE PRIVATE VARIABLE TextBox AS TextBox.
  DEFINE PRIVATE VARIABLE OkBtn   AS Button.

  CONSTRUCTOR DemoUserControl ( ):
    InitializeComponents( ).
  END CONSTRUCTOR.

  METHOD PRIVATE VOID InitializeComponents( ):
    /* Instantiate text box and button classes */
    TextBox   = NEW TextBox ( ).
    OkBtn     = NEW Button ( ).

    /* Set the text of the button, and text box */
    TextBox:Text = "Enter text here".
    OkBtn:Text   = "OK".

    /* Set the size and location of the container and controls */
    THIS-OBJECT:Size   = NEW System.Drawing.Size(200, 200).
    TextBox:Location = NEW System.Drawing.Point(15, 20).
    OkBtn:Size       = NEW System.Drawing.Size(30, 20).
    OkBtn:Location   = NEW System.Drawing.Point(125, 20).

    /* Add the text box and button controls to the container */
    THIS-OBJECT:Controls:Add(TextBox).
    THIS-OBJECT:Controls:Add(OkBtn).
  END METHOD.

END CLASS.
```

The following class instantiates this user control and adds it to a form:

```
USING System.Windows.Forms.* FROM ASSEMBLY.

CLASS DemoUserControlForm INHERITS Progress.Windows.Form:

  /* Variable for text box and button in control container */
  DEFINE PRIVATE VARIABLE rUserControl AS DemoUserControl.

  CONSTRUCTOR DemoUserControlForm ( ):
    InitializeComponents( ).
  END CONSTRUCTOR.

  /* Display and wait for this modal form to close. The caller blocks at a
     call to this method */
  METHOD PUBLIC VOID Wait ( ):
    WAIT-FOR Application:Run (THIS-OBJECT).
  END METHOD.

   METHOD PRIVATE VOID InitializeComponents( ):
    /* Instantiate user control */
    rUserControl = NEW DemoUserControl ( ).

    /* Set the text of the form title, button, and text box */
    THIS-OBJECT:Text = "This is my form".

    /* Set the location of the user control */
    rUserControl:Location = NEW System.Drawing.Point(15, 20).

    /* Add the user control to the form */
    THIS-OBJECT:Controls:Add (rUserControl).
  END METHOD.

END CLASS.
```

To instantiate `DemoUserControlForm` and display the form with the user control, you can run a procedure that contains the following code:

```
DEFINE VARIABLE rMyUIForm AS CLASS DemoUserControlForm.

rMyUIForm = NEW DemoUserControlForm().
rMyUIForm:Wait().
```

For more examples, see the topics on creating and using forms and controls in *Use GUI for .NET Objects in ABL Applications*.

Notes

- Once you have created an instance of a control container and added your existing controls to the container, you can add the container instance to a form as you would any single control. The controls included in the control container are displayed when the form is displayed.
- For more information on using this control container class, see the topics on creating and using forms and controls in *Use GUI for .NET Objects in ABL Applications*.
- For information on the public class members inherited from the `System.Windows.Forms.UserControl` class, refer to the .NET Framework class library.

See also

Progress.Windows.Form class

Progress.Windows.WindowContainer class
(Windows only; GUI for .NET only)

A control container that allows you to embed the client area of an ABL window for display in a .NET form. When added to a .NET form, this container allows the embedded ABL client area to appear as though it is added directly to the client area of the .NET form. This allows the widgets in the ABL client area to be displayed in the .NET form, but also allows you to interact with these widgets much as if they were still displayed in the original ABL window.

Serializable:

No

Constructor

```
PUBLIC WindowContainer ( )
```

Super Class

`System.Windows.Forms.UserControl` class (from the .NET Framework)

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

EmbeddedWindow property	–	–

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events (beyond those it inherits from its base class).

Notes

- To embed the client area of an ABL window in a .NET form, set the `EmbeddedWindow` property of the `WindowContainer` to the handle of the ABL window and add the `WindowContainer` to a Progress.Windows.Form class instance. For more information on how ABL supports the behavior of the ABL window and its client area embedded in a form using a `Progress.Windows.WindowContainer`, see the EmbeddedWindow property reference entry.
- You can use multiple instances of this class to embed the client areas of one or more ABL windows in the client area of any single .NET form. However, a `Progress.Windows.MDIChildForm` is designed to embed the client area of only one ABL window that you associate with the object. For more information, see the Progress.Windows.MDIChildForm class reference entry.
- The menus from an embedded ABL window are not added to any form to which you add the `WindowContainer`. You must add any associated menu or toolbar functionality to the form using corresponding .NET controls.

- For any `WindowContainer` that you add to a `Progress.Windows.Form`, you must programmatically resize and reposition the `WindowContainer` to fit the client area of the form as it is being resized.
- When you embed the client area of an ABL window in a form, the window and its widgets do not interact directly with other .NET controls that may be added to the form. This means, for example, the embedded client area does not participate in the tab order of the form. Thus, there is no way to tab into the embedded client area from another .NET control or WindowContainer, and there is no way to tab out of the embedded client area into another .NET control or `WindowContainer`. All tabbing within an embedded client area stays within the `WindowContainer` where it is embedded.
- The AVM passes all unhandled keystrokes in an embedded ABL client area to the form that contains it. This means that you can interact with menus and use menu and toolbar accelerator keys even when focus is on the embedded client area.

See also

Progress.Windows.MDIChildForm class

System.Exception class (OpenEdge-enhanced)
(Windows only; GUI for .NET only)

.NET errors are represented by error objects known as exceptions, and the .NET `System.Exception` class is the base class for all .NET `Exception` objects. To enable an ABL application to trap .NET exceptions that are raised from .NET objects, OpenEdge has enhanced `System.Exception` to implement the `Progress.Lang.Error` interface. By implementing this interface, most .NET exceptions that are raised in the ABL context can be handled by both traditional and structured ABL error handling constructs in much the same way as ABL errors.

When using structured error handling, you can catch and throw a `System.Exception` object and consult the same properties and methods as for a Progress.Lang.ProError class.

Serializable:

No

Constructors

```
Exception ( )
```

Super Class

`System.Object` class (from the .NET Framework)

Interfaces

Progress.Lang.Error interface

Public Properties

CallStack property	NumMessages property	Severity property

Public Methods

GetMessage( ) method (Class)	GetMessageNum( ) method

Public Events

This class does not contain events.

Notes

- Some .NET UI controls trap and handle errors before they get to the AVM. However, some kinds of exceptions, such as `System.AccesssViolationException`, cause `Application:Run( )` to terminate and raise a STOP condition on the executing WAIT-FOR or PROCESS EVENTS statement. You can trap this condition using the ON STOP phrase of an enclosing block. However, because `Application:Run( )` exits and all displayed forms are closed, the most that you can do is to clean-up and attempt a graceful close of your application.
- All .NET exceptions have a `Message` property from the `System.Exception` base class. In addition, if there was a chain of errors that lead to the current exception, an `InnerException` property references the most recent `Exception` object in that chain of `Exception` objects, and so on, for any number of inner exceptions. Each inner exception, then, is just another `Exception` object (or something derived from it), with its own `Message` property. In addition, specific types of `Exception` objects have additional information available from custom properties. For example, the `FileNotFoundException` has a `FileName` property, which is the name of the file that cannot be found.

  Your ABL application can access all of the messages in the .NET `Exception` object (i.e., the message from the object itself plus any from inner exceptions) using the methods of `Progress.Lang.Error`. However, if you want to access any additional information such as the `FileName` custom property of the `FileNotFoundException` object, you must do so using the native properties and methods of the .NET `Exception` object itself.

See also

PROCESS EVENTS statement, Progress.Lang.Error interface, WAIT-FOR statement (.NET and ABL)

9

Class Properties and Methods Reference

This section contains reference entries that describe each built-in class property and method that ABL supports for working with ABL classes and structured error handling. Class properties and methods are mechanisms that allow you to monitor and control the behavior of class-based objects. The class properties and methods described in this section are provided by two types of OpenEdge classes:

- Built-in ABL classes that support working with user-defined ABL classes
- Built-in .NET classes that support working with .NET forms and controls

Some methods of built-in ABL classes also support .NET features, especially for handling .NET exceptions using ABL structured error handling.

For information on:

- The built-in classes provided by OpenEdge for working with both ABL and .NET classes (including structured error handling), see the Class, Interface, and Enumeration Reference.
- Accessing class properties in ABL, see the Class-based property access reference entry.
- Calling class methods in ABL, see the Class-based method call reference entry.

Syntax for describing class method parameter and property data types

The syntax to describe the parameters of class methods in this section uses a variation of the ABL Parameter definition syntax, which includes the parameter mode and an AS option to indicate the ABL data type of the parameter. For example:

```
INPUT value AS DECIMAL
```

Note: This is similar to the parameter definition syntax used to list method prototypes in the Class Browser.

A similar syntax is used to describe the parameters of .NET methods, with an optional additional notation to indicate the corresponding .NET data type. You must know the explicit .NET data type of a .NET method parameter:

- To call an overloaded .NET method (see the Parameter passing syntax reference entry)
- When you override an inherited .NET method (see the METHOD statement reference entry)
- When you implement a method of a .NET interface (see the METHOD statement reference entry)

The ABL data type denoted by the AS option indicates how ABL treats the .NET parameter at run time. The parameter description uses the additional notation when the actual .NET data type of the parameter is a .NET mapped type other than the default match for the specified ABL data type. For information on matching .NET mapped types to ABL data types, see the Data types reference entry. If the .NET data type is not a mapped type or it is the default match for the specified ABL data type, this additional notation does not appear.

By convention throughout this section, when a particular parameter definition requires the additional notation, the syntax specifies the corresponding ABL data type and it explicitly the .NET data type mapping using an additional ABL keyword (*AS data type*) in parentheses (see Table 25: Explicit mappings between ABL and .NET data types. For example, the syntax for a .NET INPUT parameter that takes a single-precision floating-point number is represented as follows:

```
INPUT value AS DECIMAL (FLOAT)
```

Thus, the AS option for the ABL parameter definition syntax indicates the ABL DECIMAL data type, and the AS data type keyword FLOAT, which indicates an explicit .NET data type mapping to System.Single.

A similar notation is also used to document the data types of .NET method return values and .NET properties.

For more comprehensive information on mapping .NET data types to ABL data types, see *Use GUI for .NET Objects in ABL Applications*.

AccessMode property

Returns an instance of `Progress.Reflect.AccessMode` that indicates the access mode for the constructor, event, method, data member, or property,

Data type: Progress.Reflect.AccessMode enumeration

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Constructor class, Progress.Reflect.Event class, Progress.Reflect.Method class, Progress.Reflect.Property class, Progress.Reflect.Variable class

When applied to a property of a class, the `AccessMode` property returns the access mode for the property itself, not for the property's GET and SET accessors. To provide that information, the `Progress.Reflect.Property` class has two additional properties, `GetterAccessMode` and `SetterAccessMode`.

See also

GetterAccessMode property, SetterAccessMode property

AccessDenied property

An authentication status code that indicates authentication failure because of a denied access attempt.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

AccessRevoked property

An authentication status code that indicates authentication failure because access has been revoked.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

AccountDisabled property

An authentication status code that indicates authentication failed because the user account or domain has been disabled.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

AccountExpired property

An authentication status code that indicates authentication failed because the user account has expired.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

ActBufferId property

The `ActBufferId` property identifies the `ActBuffer` Virtual System Table (VST). This VST contains three records that provide data on the activity of the temp-table database buffer cache. The first and second records are identical (total and primary buffer pool). The third record contains data on the alternate buffer pool, which is not available for the temp-table database. You can pass the `ActBufferId` property to the `GetVSTHandle()`method to return the handle to the `ActBuffer` VST. This property is read-only.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

ActIndexId property

The `ActIndexId` property identifies the `ActIndex` VST. This VST contains index activity data such as the number of entry finds, creates, and deletes; the number of locked entries removed; and the numbers of split and free blocks. You can pass the `ActIndexId` property to the `GetVSTHandle()` method to return the handle to the `ActIndex` VST. This property is read-only.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

ActIOFileId property

The `ActIOFileId` property identifies the `ActIOFile` VST. This VST contains input/output activity data, such as the number of reads; writes; and extends for each file. You can pass the `ActIOFileId` property to the `GetVSTHandle()` method to return the handle to the `ActIOFile` VST. This property is read-only.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

ActIOTypeId property

The `ActIOTypeId` property identifies the `ActIOType` VST. This VST contains data about types of input/output activity, such as database reads and writes; before-image and after-image reads; total reads; before-image and after-image writes; committed transactions. You can pass the `ActIOTypeId` property to the `GetVSTHandle()` method to return the handle to the `ActIOType` VST.This property is read-only.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

ActOtherId property

The `ActOtherId` property identifies the `ActOther` VST. This VST displays miscellaneous activity, such as the number of commits; undo operations; semaphore waits; master block flushes; and database up time. You can pass the `ActOtherId` property to the `GetVSTHandle()` method to return the handle to the `ActOther` VST. This property is read-only.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

ActRecordId property

The `ActRecordId` property identifies the `ActRecord` VST. This VST contains record activity data, such as the number of bytes created; deleted; read, and updated; the number of fragments created, deleted, read, and updated; the number of records created; deleted; read; and updated; the number of transactions committed. You can pass the `ActRecordId` property to the `GetVSTHandle()` method to return the handle to the `ActRecord` VST. This property is read-only.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

ActSpaceId property

The `ActSpaceId` property identifies the `ActSpace` VST. This VST displays space allocation information, such as the number of temp-table database extends; the number of times a block was used from and returned to the free chain; the number of times space was allocated for a record (from the rm chain or from the free chain); the number of bytes allocated for record fragments, the number of rm blocks examined or removed, the number of blocks added to the front or back of the rm chain, the number of moved blocks, the number of locked chain entries, the number of transactions committed, and database up time. You can pass the `ActSpaceId` property to the `GetVSTHandle()` method to return the handle to the `ActSpace` VST. This property is read-only.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

ActSummaryId property

The `ActSummaryId` property identifies the `ActSummary` VST. This VST contains general information about temp-table database activity, such as the number of transactions committed and rolled back; the number of records read, updated, created, and deleted; the number of record locks and waits; the number of database reads and writes; before-image and after-image information; and buffer information. You can pass the `ActSummaryId` property to the `GetVSTHandle()` method to return the handle to the `ActSummary` VST. This property is read-only.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

AdapterType property

Returns a Progress.ApplicationServer.AdapterTypes instance that identifies the transport for the current request running on a Progress Application Server (PAS). If not running on a PAS, it returns the Unknown value (?).

Data type: Progress.ApplicationServer.AdapterTypes enumeration

Access: PUBLIC Read-only

Applies to: Progress.Lang.OERequestInfo class

Add( ) method (JsonArray)

Creates one or more new elements and adds them at a specified index or to the end of the JsonArray. On successful completion, this method returns the index of the last newly added element.

Return type: INTEGER

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
Add( INPUT value AS CHARACTER )

Add( INPUT array-value AS CHARACTER EXTENT )

Add( INPUT value AS COM-HANDLE )

Add( INPUT array-value AS COM-HANDLE EXTENT )

Add( INPUT value AS DATE )

Add( INPUT array-value AS DATE EXTENT )

Add( INPUT value AS DATETIME )

Add( INPUT array-value AS DATETIME EXTENT )
```

```
Add( INPUT value AS DATETIME-TZ )
Add( INPUT array-value AS DATETIME-TZ EXTENT )
Add( INPUT value AS DECIMAL )
Add( INPUT array-value AS DECIMAL EXTENT )
Add( INPUT value AS HANDLE )
Add( INPUT array-value AS HANDLE EXTENT )
Add( INPUT value AS INT64 )
Add( INPUT array-value AS INT64 EXTENT )
Add( INPUT value AS INTEGER )
Add( INPUT array-value AS INTEGER EXTENT )
Add( INPUT value AS LOGICAL )
Add( INPUT array-value AS LOGICAL EXTENT )
Add( INPUT value AS LONGCHAR )
Add( INPUT array-value AS LONGCHAR EXTENT )
Add( INPUT value AS MEMPTR )
Add( INPUT value AS CHARACTER )
Add( INPUT array-value AS CHARACTER EXTENT )
Add( INPUT value AS COM-HANDLE )
Add( INPUT array-value AS COM-HANDLE EXTENT )
Add( INPUT value AS DATE )
Add( INPUT array-value AS DATE EXTENT )
Add( INPUT value AS DATETIME )
Add( INPUT array-value AS DATETIME EXTENT )
Add( INPUT value AS DATETIME-TZ )
Add( INPUT array-value AS DATETIME-TZ EXTENT )
Add( INPUT value AS DECIMAL )
Add( INPUT array-value AS DECIMAL EXTENT )
Add( INPUT value AS HANDLE )
Add( INPUT array-value AS HANDLE EXTENT )
Add( INPUT value AS INT64 )
Add( INPUT array-value AS INT64 EXTENT )
Add( INPUT value AS INTEGER )
Add( INPUT array-value AS INTEGER EXTENT )
```

```
Add( INPUT value AS LOGICAL )

Add( INPUT array-value AS LOGICAL EXTENT )

Add( INPUT value AS LONGCHAR )

Add( INPUT array-value AS LONGCHAR EXTENT )

Add( INPUT value AS MEMPTR )

Add( INPUT array-value AS MEMPTR EXTENT )

Add( INPUT value AS RAW )

Add( INPUT array-value AS RAW EXTENT )

Add( INPUT value AS RECID )

Add( INPUT array-value AS RECID EXTENT )

Add( INPUT value AS ROWID )

Add( INPUT array-value AS ROWID EXTENT )

Add( INPUT value AS CLASS JsonObject )

Add( INPUT array-value AS CLASS JsonObject EXTENT )

Add( INPUT value AS CLASS JsonArray )

Add( INPUT array-value AS CLASS JsonArray EXTENT )

Add( INPUT index AS INTEGER, INPUT value AS CHARACTER )

Add( INPUT index AS INTEGER,
     INPUT array-value AS CHARACTER EXTENT )

Add( INPUT index AS INTEGER, INPUT value AS COM-HANDLE )

Add( INPUT index AS INTEGER,
     INPUT array-value AS COM-HANDLE EXTENT )

Add( INPUT index AS INTEGER, INPUT value AS DATE )

Add( INPUT index AS INTEGER,
     INPUT array-value AS DATE EXTENT )

Add( INPUT index AS INTEGER, INPUT value AS DATETIME )

Add( INPUT index AS INTEGER,
     INPUT array-value AS DATETIME EXTENT )

Add( INPUT index AS INTEGER, INPUT value AS DATETIME-TZ )

Add( INPUT index AS INTEGER,
     INPUT array-value AS DATETIME-TZ EXTENT )

Add( INPUT index AS INTEGER, INPUT value AS DECIMAL )

Add( INPUT index AS INTEGER,
     INPUT array-value AS DECIMAL EXTENT )

Add( INPUT index AS INTEGER, INPUT value AS HANDLE )

Add( INPUT index AS INTEGER,
```

```
     INPUT array-value AS HANDLE EXTENT )

Add( INPUT index AS INTEGER, INPUT value AS INT64 )

Add( INPUT index AS INTEGER,
     INPUT array-value AS INT64 EXTENT )

Add( INPUT index AS INTEGER, INPUT value AS INTEGER )

Add( INPUT index AS INTEGER,
     INPUT array-value AS INTEGER EXTENT )

Add( INPUT index AS INTEGER, INPUT value AS LOGICAL )

Add( INPUT index AS INTEGER,
     INPUT array-value AS LOGICAL EXTENT )

Add( INPUT index AS INTEGER, INPUT value AS LONGCHAR )

Add( INPUT index AS INTEGER,
     INPUT array-value AS LONGCHAR EXTENT )

Add( INPUT index AS INTEGER, INPUT value AS MEMPTR )

Add( INPUT index AS INTEGER,
     INPUT array-value AS MEMPTR EXTENT )

Add( INPUT index AS INTEGER, INPUT value AS RAW )

Add( INPUT index AS INTEGER,
     INPUT array-value AS RAW EXTENT )

Add( INPUT index AS INTEGER, INPUT value AS RECID )

Add( INPUT index AS INTEGER,
     INPUT array-value AS RECID EXTENT )

Add( INPUT index AS INTEGER, INPUT value AS ROWID )

Add( INPUT index AS INTEGER,
     INPUT array-value AS ROWID EXTENT )

Add( INPUT index AS INTEGER,
     INPUT value AS CLASS JsonObject )

Add( INPUT index AS INTEGER,
     INPUT array-value AS CLASS JsonObject EXTENT )

Add( INPUT index AS INTEGER,
     INPUT value AS CLASS JsonArray )

Add( INPUT index AS INTEGER,
     INPUT array-value AS CLASS JsonArray EXTENT )
```

value

The value to which the new element is to be set. If *value* is the Unknown value (?), the element is set to the JSON null value.

If *value* is a simple type, the JsonArray:Length property is incremented by one. If *value* is an ABL array, JsonArray:Length is incremented by the length of the ABL array.

array-value

ABL array containing values used to initialize a set of new elements. Each element of the ABL array is used to initialize one new element in the JsonArray. Any Unknown value (?) within the ABL array will result in a JSON null value in the JsonArray. If *array-value* itself is the Unknown value (?) or indeterminate, a JsonError is raised.

index

An INTEGER identifying the element after which the new element or elements are added. Indexing into JsonArrays is 1-based. For example, myArray:Add(5, "custnum") inserts a String value "custnum" as element 6 into the JsonArray. All elements starting with the original sixth element are shifted by one position.

If *index* is 0, the new element is inserted at the beginning of the array making that new element's index 1. If *index* is not provided, the default behavior is to add new elements to the end of the array. If *index* is the Unknown value (?), less than 0, or is greater than the length of the array, a JsonError is raised.

The data type of the added JSON value set by this method call depends upon the ABL data type of the *value* parameter.

Table 132: Value parameter

A value parameter of data type	Data type of JSON value
CHARACTER, LONGCHAR	string
INTEGER, INT64, DECIMAL	number
LOGICAL	boolean
MEMPTR, RAW, ROWID	string with a value as if you had called BASE64-ENCODE( ) on *value*
DATE, DATETIME, DATETIME-TZ	string with a value as if you had called ISO-DATE( ) on *value*
COM-HANDLE, HANDLE, RECID	number with a value as if you had called INTEGER( ) on *value*

The following example demonstrates how the Add( ) overloaded methods affect a JsonArray:

```
DEFINE VARIABLE a AS INTEGER EXTENT 3 INITIAL [3, 2, 1].
DEFINE VARIABLE b AS CHARACTER EXTENT 2 INITIAL ["Hello", "world"].
DEFINE VARIABLE c AS DATE EXTENT 2 INITIAL ["9/20/10", "10/10/10"].

myArr = NEW JsonArray(a)
/* myArr is now [3, 2, 1] */

myArr:Add(TRUE).
/* myArr is now [3, 2, 1, true] */
myArr:Add(b).
/* myArr  is now [3, 2, 1, true, "Hello", "world"] */

myArr:Add(2, c).
/* myArr is now [3, 2, "2010-09-20", "2010-10-10", 1, true, "Hello", "world"]
*/
```

Add( ) method (JsonObject)

Creates a property with a given name and value. On successful execution, this method returns TRUE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
Add( INPUT property-name AS CHARACTER,
     INPUT value AS CHARACTER )

Add( INPUT property-name AS CHARACTER,
     INPUT value AS COM-HANDLE )

Add( INPUT property-name AS CHARACTER,
     INPUT value AS DATE )

Add( INPUT property-name AS CHARACTER,
     INPUT value AS DATETIME )

Add( INPUT property-name AS CHARACTER,
     INPUT value AS DATETIME-TZ )

Add( INPUT property-name AS CHARACTER,
     INPUT value AS DECIMAL )

Add( INPUT property-name AS CHARACTER,
     INPUT value AS HANDLE )

Add( INPUT property-name AS CHARACTER,
     INPUT value AS INT64 )

Add( INPUT property-name AS CHARACTER,
     INPUT value AS INTEGER )

Add( INPUT property-name AS CHARACTER,
     INPUT value AS LOGICAL )
```

```
Add( INPUT property-name AS CHARACTER,
     INPUT value AS LONGCHAR )

Add( INPUT property-name AS CHARACTER,
     INPUT value AS MEMPTR )

Add( INPUT property-name AS CHARACTER,
     INPUT value AS RAW )

Add( INPUT property-name AS CHARACTER,
     INPUT value AS RECID )

Add( INPUT property-name AS CHARACTER,
     INPUT value AS ROWID )

Add( INPUT property-name AS CHARACTER,
     INPUT value AS CLASS JsonObject )

Add( INPUT property-name AS CHARACTER,
     INPUT value AS CLASS JsonArray )
```

property-name

A CHARACTER expression naming the new property to be created. This name can be any string that can be represented as a valid Unicode string.

If a property by this name already exists in this JsonObject or if this parameter is an empty string ("") or is the Unknown value (?), a JsonError is raised.

property-name is case-sensitive.

value

The value to which the new property is to be set. The data type of the property is defined by the data type mapping described below. If *value* is the Unknown value (?) the property is set to the a JSON null value.

If you add a JsonObject or JsonArray that references this JsonObject or a construct further up the JSON construct tree, a JsonError is raised.

The data type of the added JSON value set by this method call depends upon the ABL data type of the *value* parameter.

Table 133: Value parameter

A value parameter of data type	Data type of JSON value
CHARACTER, LONGCHAR	string
INTEGER, INT64, DECIMAL	number
LOGICAL	boolean
MEMPTR, RAW, ROWID	string with a value as if you had called BASE64-ENCODE( ) on *value*

A value parameter of data type	Data type of JSON value
DATE, DATETIME, DATETIME-TZ	string with a value as if you had called `ISO-DATE( )` on *`value`*
COM-HANDLE, HANDLE, RECID	number with a value as if you had called `INTEGER( )` on *`value`*

Add( ) method (Progress.Util.EnumHelper)
(Windows only; GUI for .NET only)

Adds the underlying values of the specified enumeration types and returns the sum as a new instance.

Return type: System.Enum class (from the .NET Framework)

Access: PUBLIC STATIC

Applies to: Progress.Util.EnumHelper class

Syntax

```
EnumHelper:Add ( INPUT enum1 AS CLASS System.Enum,
                 INPUT enum2 AS CLASS System.Enum )
```

enum1

Static reference to an enumeration value, or a reference to an enumeration type instance.

enum2

Static reference to an enumeration value, or a reference to an enumeration type instance.

Specify a static reference to an enumeration value using the following syntax:

```
enum-type:enum-member
```

Where *enum-type* is the type name of the enumeration and *enum-member* is a member name of the enumeration. For example:

```
System.Windows.Forms.AnchorStyles:Bottom
```

If you want to use the returned System.Enum instance as a different enumeration type, you must cast the instance to that specific type using the CAST function.

For information on the System.Enum class and .NET enumeration types, refer to the .NET Framework class library.

See also

CAST function

AddMessage( ) method

Adds a message to the AppError object with the values from the *ErrorMessage* and *MessageNumber* arguments to the end of the message list. Your application provides the message number and text. Access error messages and message numbers with the GetMessage( ) and GetMessageNum( ) methods. This method increments the NumMessages property on the AppError by 1.

Return type: VOID

Access: PUBLIC

Applies to: Progress.Lang.AppError class

Syntax

```
AddMessage ( INPUT ErrorMessage  AS CHARACTER,
             INPUT MessageNumber AS INTEGER )
```

AddNull( ) method (JsonArray)

Creates one or more new elements with a null value and adds them at a specified index or to the end of the JsonArray.

This method is useful in situations where you know at compile time to add JSON null values. To use one of the Add( ) overloadings, you must set a variable to the Unknown value (?) first, then call Add( ) with the variable. Using the Unknown value (?) value as a constant in one of the Add( ) overloadings causes a compilation error because the compiler is unable to resolve the overloading to be used.

On successful completion, this method returns the index of the last newly added element.

Return type: INTEGER

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
AddNull ( )

AddNull ( INPUT count AS INTEGER )

AddNull ( INPUT index AS INTEGER ,
          INPUT count AS INTEGER )
```

With no parameters present in the method, one JSON null element is added to the end of the JsonArray and the JsonArray:Length property is incremented by 1.

count

An INTEGER indicating the number of elements to add to the JsonArray. Each element is initialized to the JSON null value. JsonArray:Length is incremented by *count*.

If *count* is less than 1, or is the Unknown value (?), a JsonError is raised.

index

An INTEGER identifying the element after which the new element or elements are added. Indexing into JsonArrays is 1-based. For example, myArray:AddNull(5, 3) inserts three JSON null values as elements 6, 7 and 8 into the JsonArray. All elements starting with the original sixth element are shifted by three positions.

If the index is 0, the new element is inserted at the beginning of the array making that new elements index 1. If the index is not provided, the default behavior is to add new elements to the end of the array.

If index is the Unknown value (?), or less than 0, or greater than the length of the array, a JsonError is raised.

AddNull( ) method (JsonObject)

Creates a property with a given name and sets its value to null.

This method is useful for those situations where you know at compile time that you will be adding a JSON null value. To use one of the Add( ) overloadings, you must set a variable to the Unknown value (?) first, then call Add( ) with the variable. Using the Unknown value (?) value as a constant in one of the Add( ) overloadings causes a compilation error, because the compiler is unable to resolve the overloading to use.

On successful execution, this method returns TRUE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
AddNull( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression naming the new property to be created. This name may be any string that can be represented as a valid Unicode string.

If *property-name* is the empty string (""), is the Unknown value (?), or does not exist, a JsonError is raised.

property-name is case-sensitive.

AddNumber( ) method (JsonArray)

Creates one or more new elements, setting each element's value to a JSON number value represented by the CHARACTER parameter and adds it to the JsonArray.

This method is useful in situations where none of the ABL numeric data types is used to hold a numeric value. For example, 10E+100.

On successful execution, this method returns the index of the newly added element. If the value is an ABL array, the index of the last element added is returned.

Return type: INTEGER

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
AddNumber( INPUT value AS CHARACTER )

AddNumber( INPUT array-value AS CHARACTER EXTENT )

AddNumber( INPUT index AS INTEGER,
           INPUT value AS CHARACTER )

AddNumber( INPUT index AS INTEGER,
           INPUT array-value AS CHARACTER EXTENT )
```

When this method is called with one parameter, the new elements are added to the end of the JsonArray. When this method is called with two parameters, the AVM inserts the new elements after the element at the specified index.

If the overloading with an ABL array parameter is used, the AVM uses each element of the ABL array to initialize a new element of the JsonArray.

value

A CHARACTER expression representing a numeric value to which the new element is to be set. The value must match the pattern for valid JSON number values. These may be integers ([-]dddd), decimals ([-]ddd.ddd), or scientific notation, ([-]ddd[.ddd]e[+|-]ddd). If the value is the Unknown value (?), the element is set to the a JSON null value.

If the value is not a valid JSON number value or is an empty string (""), a JsonError is raised.

array-value

A CHARACTER array containing values used to initialize a set of new elements. Each element of the ABL array is used to initialize one new element in the JsonArray. The JsonArray grows by as many elements as are in the ABL array. The value must match the pattern for valid JSON number values. These may be integers ([-]dddd), decimals ([-]ddd.ddd), or scientific notation ([-]ddd[.ddd]e[+|-]ddd). Any Unknown value (?) within the ABL array will result in a JSON null value in the JsonArray.

A JsonError is raised if:

- *array-value* is the Unknown value (?) or indeterminate.
- any element in the ABL array is not a valid JSON number value.

index

An INTEGER identifying the element after which the new element or elements are added. Indexing into JsonArrays is 1-based. For example, `myArray:AddNumber(8, "10e15")` inserts a JSON number value as element 9 into the JsonArray. All elements starting with the original ninth element will be shifted by one position.

If *index* is 0, the new element is inserted at the beginning of the array making that new elements index 1.

If *index* is the Unknown value (?), less than 0, or is larger than the length of the array, a JsonError is raised.

AddNumber( ) method (JsonObject)

Creates a property with a given name and JSON number value. This method is useful in situations where none of the ABL numeric data types can be used to hold a JSON numeric value. For example, 10E+100.

On successful execution, this method returns TRUE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
AddNumber( INPUT property-name AS CHARACTER,
           INPUT value AS CHARACTER )
```

property-name

A CHARACTER expression representing the new property to be created. This name can be any string that can be represented as a valid Unicode string.

If *property-name* is the empty string ("") , is the Unknown value (?), or if the property already exists, a JsonError is raised.

property-name is case-sensitive.

value

A CHARACTER expression representing a numeric value to which the new element is to be set. The value must match the pattern for valid JSON number values. These may be integers ([-]dddd), decimals ([-]ddd.ddd), or scientific notation ([-]ddd[.ddd]e[+|-]ddd). If *value* is the Unknown value (?), the element is set to the a JSON null value. If the value is not a valid JSON number value or is an empty string (""), a JsonError is raised.

AgentId property
(PAS for OE only)

Identifies the Progress Application Server agent associated with a given request or response. If there is no valid response or request running in the session, the property defaults to the Unknown value (?).

Data type: INTEGER

Access: PUBLIC Read-only

Applies to: Progress.Lang.OERequestInfo class

AllowEdit property
(Windows only; GUI for .NET only)

Indicates whether the .NET control should allow the user to edit values in the bound ABL data source object. The default value is TRUE.

You can use this property when the BindingSource object is bound to a query or a buffer. When bound to a ProDataSet object, this property applies only to the top-level table displayed in the .NET control.

Data type: LOGICAL

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

Use this property only with .NET controls that support an edit operation.

See also

ChildAllowEdit property

AllowNew property
(Windows only; GUI for .NET only)

Indicates whether the .NET control should allow the user to add new records to the bound ABL data source object. The default value is TRUE.

You can use this property when the BindingSource object is bound to a query. When bound to a ProDataSet object, this property applies only to the top-level table displayed in the .NET control.

Data type: LOGICAL

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

Use this property only with .NET controls that support an add operation.

See also

ChildAllowNew property

AllowRemove property
(Windows only; GUI for .NET only)

Indicates whether the .NET control should allow the user to remove records from the bound ABL data source object. The default value is TRUE.

You can use this property when the BindingSource object is bound to a query. When bound to a ProDataSet object, this property applies only to the top-level table displayed in the .NET control.

Data type: LOGICAL

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

Use this property only with .NET controls that support a remove operation.

See also

ChildAllowRemove property

And( ) method
(Windows only; GUI for .NET only)

Performs a bitwise AND operation on the underlying values of the specified enumeration types and returns a new instance. You typically use this method to check the state of a flag.

Return type: System.Enum class (from the .NET Framework)

Access: PUBLIC STATIC

Applies to: Progress.Util.EnumHelper class

Syntax

```
EnumHelper:And ( INPUT enum1 AS CLASS System.Enum,
                 INPUT enum2 AS CLASS System.Enum )
```

enum1

Static reference to an enumeration value, or a reference to an enumeration type instance.

enum2

Static reference to an enumeration value, or a reference to an enumeration type instance.

Specify a static reference to an enumeration value using the following syntax:

```
enum-type:enum-member
```

Where *enum-type* is the type name of the enumeration and *enum-member* is a member name of the enumeration. For example:

```
System.Windows.Forms.AnchorStyles:Bottom
```

The following example uses this method to check the state of a flag:

```
USING System.Windows.Forms.* FROM ASSEMBLY.
USING Progress.Util.* FROM ASSEMBLY.

DEFINE VARIABLE rMyStyle AS System.Enum NO-UNDO.

/* Adds Bottom and Right to the list */
rMyStyle = EnumHelper:Or( AnchorStyles:Bottom, AnchorStyles:Right ).
DISPLAY STRING( rMyStyle ) FORMAT "x(20)" SKIP.
/* Displays "Bottom, Right" */

/* Test to see if a flag is set */
rMyStyle = EnumHelper:And( rMyStyle, AnchorStyles:Right ).
DISPLAY STRING( rMyStyle ) FORMAT "x(20)" SKIP.
/* Displays "Right" */

rMyStyle = EnumHelper:And( rMyStyle, AnchorStyles:Left ).
DISPLAY STRING( rMyStyle ) FORMAT "x(20)" SKIP.
/* Displays "None" */
```

If you want to use the returned System.Enum instance as a different enumeration type, you must cast the instance to that specific type using the CAST function.

For information on the System.Enum class and .NET enumeration types, refer to the .NET Framework class library.

See also

CAST function

Append property

Indicates whether data can be appended to the file output stream. If the property is not set by the constructor, the default value is FALSE.

Data type: LOGICAL

Access: PUBLIC Read-only

Applies to: Progress.IO.FileOutputStream class

ArchiveIndexStatistics property

When `ArchiveIndexStatistics` property is set to TRUE, it creates a dynamic temp-table `_IndexStatHistory` that stores the index statistics for every temp-table in the session range. The temp-table to which the data is archived cannot be deleted. It contains one record for every deleted application temp-table This is a read-write property.

You cannot archive the statistics when:

- `ArchiveIndexStatistics` property is set to FALSE.
- The temp-table gets deleted.

You can access the archived data through the handle returned by the `GetIndexStatHistoryHandle()` method. If you do not specify a value greater than zero for `-ttindexrangesize` parameter, an error is raised when the `ArchiveIndexStatistics` property is set to TRUE.

Data type: LOGICAL

Access: PUBLIC STATIC

Applies to: Progress.Database.TempTableInfo class

ArchiveTableStatistics property

When `ArchiveTableStatistics` property is set to TRUE, it creates a dynamic temp-table `_TableStatHistory` that stores the table statistics for every temp-table in the session range. The temp-table to which the data is archived cannot be deleted. It contains one record for every deleted application temp-table. This is a read-write property.

You cannot archive the table statistics when:

- `ArchiveTableStatistics` property is set to FALSE.
- The temp-table gets deleted.

You can access the archived data through the handle returned by the `GetTableStatHistoryHandle()` method. If you do not specify a value greater than zero for `-tttablerangesize` parameter, an error is raised when `ArchiveTableStatistics` property is set to TRUE.

Data type: LOGICAL

Access: PUBLIC STATIC

Applies to: Progress.Database.TempTableInfo class

AreEqual( ) method
(Windows only; GUI for .NET only)

Compares the underlying values of the specified enumeration types and returns TRUE if the values are equal. Otherwise, it returns FALSE.

Return type: LOGICAL

Access: PUBLIC STATIC

Applies to: Progress.Util.EnumHelper class

Syntax

```
EnumHelper:AreEqual ( INPUT enum1 AS CLASS System.Enum,
                      INPUT enum2 AS CLASS System.Enum )
```

enum1

Static reference to an enumeration value, or a reference to an enumeration type instance.

enum2

Static reference to an enumeration value, or a reference to an enumeration type instance.

Specify a static reference to an enumeration value using the following syntax:

```
enum-type:enum-member
```

Where *enum-type* is the type name of the enumeration and *enum-member* is a member name of the enumeration. For example:

```
System.Windows.Forms.AnchorStyles:Bottom
```

See also

AreNotEqual( ) method

AreNotEqual( ) method
(Windows only; GUI for .NET only)

Compares the underlying values of the specified enumeration types and returns TRUE if the values are not equal. Otherwise, it returns FALSE.

Return type: LOGICAL

Access: PUBLIC STATIC

Applies to: Progress.Util.EnumHelper class

Syntax

```
EnumHelper:AreNotEqual ( INPUT enum1 AS CLASS System.Enum,
                         INPUT enum2 AS CLASS System.Enum )
```

enum1

Static reference to an enumeration value, or a reference to an enumeration type instance.

enum2

Static reference to an enumeration value, or a reference to an enumeration type instance.

Specify a static reference to an enumeration value using the following syntax:

```
enum-type:enum-member
```

Where *enum-type* is the type name of the enumeration and *enum-member* is a member name of the enumeration. For example:

```
System.Windows.Forms.AnchorStyles:Bottom
```

See also

AreEqual( ) method

Array property

Returned by the GetType method on a JsonObject or JsonArray indicating that the JSON data type for the specified property or element is a JSON Object.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Json.ObjectModel.JsonDataType class

ArrayIndex property
(Windows only; GUI for .NET only)

The 1-based index position in an array field identified by the FieldIndex and FieldName properties for a SortRequestEventArgs object instance that indicates the array element on which to sort. If the field is not an array field, the value will be zero.

Data type: INTEGER

Access: PUBLIC Read-only

Applies to: Progress.Data.SortRequestEventArgs class

See also

Ascending property, FieldIndex property, FieldName property, SortRequest event

---

Ascending property
(Windows only; GUI for .NET only)

Indicates the order in which to sort the records in the ABL data source object. TRUE indicates ascending sort order. FALSE indicates descending sort order.

Data type: LOGICAL

Access: PUBLIC Read-only

Applies to: Progress.Data.SortRequestEventArgs class

See also

Array property, FieldIndex property, FieldName property, SortRequest event

Assign( ) method (Progress.Data.BindingSource)
(Windows only; GUI for .NET only)

Assigns input values from the current row in the .NET control back to the corresponding record in the bound ABL data source object buffer. When a control is bound to a single field, the BindingSource object assigns the input value to the corresponding field in the buffer.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Data.BindingSource class

Syntax

```
Assign( )
```

You can only use this method for validation at the row level. When a control fires a row-level event, the BindingSource object has access to the input values for all fields in the row. When a control fires a field-level event, the BindingSource object might not have access to that field's input value.

If the method assigns values successfully, it returns TRUE; otherwise, it returns FALSE.

AuthenticationAccess property

An authentication status code that indicates authentication failed because user account information is not accessible.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

AuthenticationFailed property

An authentication status code that indicates a non-specific authentication failure, which may be the result of a user account being disabled or some other user login restriction.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

AutoSort property
(Windows only; GUI for .NET only)

Indicates whether the BindingSource object automatically sorts records in the ABL data source object when some user action in the bound .NET control generates a sort operation (such as clicking on a column header in a grid control).

You can use this property when the BindingSource object is bound to a query. If bound to a ProDataSet object, this property applies only to the top-level table displayed in the .NET control.

This property applies only to .NET controls that rely on their data source for sorting.

Data type: LOGICAL

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

The query must have been opened with the QUERY-PREPARE and QUERY-OPEN methods (not the OPEN QUERY statement). Otherwise, the AVM generates a run-time error.

Note: Progress Software Corporation recommends that you open the query using the PRESELECT option, which optimizes query performance.

When TRUE, and the user initiates a sort request, the BindingSource object reopens the query associated with the ABL data source object with an appropriate BY phrase, which resorts the data in the ABL data source object. The BindingSource object replaces any existing BY phrase in the query. For more information about the BY phrase, see the OPEN QUERY statement reference entry.

When FALSE, the BindingSource object raises a SortRequest event that the application can trap and handle. If the application does not handle the event, the data in the ABL data source object is not resorted. The default value is FALSE.

If record batching is enabled (that is, the Batching property is TRUE), the BindingSource object resorts only the records in the current result set.

See also

Batching property, SortRequest event

AutoSync property
(Windows only; GUI for .NET only)

Indicates whether the BindingSource object automatically synchronizes (refreshes) all data displayed in any bound .NET control after one of the following ABL operations on the bound ABL data source object occurs:

- Reopening the query associated with the data source using either the QUERY-PREPARE and QUERY-OPEN methods or the OPEN QUERY statement.
- Repositioning the query associated with the data source using either the REPOSITION statement or any of the REPOSITION methods.

You can use this property when the BindingSource object is bound to a query or a buffer. If bound to a ProDataSet object, this property applies only to the top-level table displayed in the .NET control.

Data type: LOGICAL

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

When TRUE, the BindingSource object automatically synchronizes the data displayed in any bound .NET control. When FALSE, the application must handle this synchronization. The default value is TRUE.

Invoking the CREATE-RESULT-LIST-ENTRY( ) method or the DELETE-RESULT-LIST-ENTRY( ) method to update the result list does not cause the BindingSource object to automatically refresh the data displayed in the bound .NET controls. In this case, you can either invoke the RefreshAll( ) method or reopen the query (unless otherwise indicated).

For more information about synchronizing data, see the topics on binding ABL data to .NET controls in *Use GUI for .NET Objects in ABL Applications*.

See also

RefreshAll( ) method

AutoUpdate property
(Windows only; GUI for .NET only)

Indicates whether the BindingSource object automatically updates records in the ABL data source object when some user action in the bound .NET control generates an update operation (such as adding, removing, or editing fields).

Caution: This property is intended only for rapid prototyping purposes. The BindingSource object cannot perform all data-handling tasks automatically (for example, validating data, updating related data, and creating or deleting records in a join query). Also, if any error occurs while updating the ABL data source object, the BindingSource object cannot always notify the .NET control and the control might not behave as expected. The application must take this into account when automatically applying updates in the ABL data source object. For more information, see the data-binding topics in *Use GUI for .NET Objects in ABL Applications*.

Data type: LOGICAL

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

When TRUE, the BindingSource object automatically applies updates made to any records in the ABL data source object. The default value is FALSE.

AvailableBytes( ) method

Returns the number of bytes remaining for a read.

Return type: INT64

Access: PUBLIC

Applies to: Progress.IO.MemoryInputStream class

Syntax

```
AvailableBytes( )
```

BandIndex property
(Windows only; GUI for .NET only)

Indicates the band in a hierarchical control which uses the current record. A *band* consists of all the records at a given level in the hierarchical display. A given band might contain sets of child rows from different parent records, as shown:

```
Band 0
    Band 1
        Band 2
        Band 2
            Band 3
            Band 3
    Band 1

Band 0
    Band 1
    Band 1

Band 0
    Band 1
        Band 2
            Band 3
            Band 3
    Band 1
        Band 2
        Band 2
    Band 1
    Band 1
        Band 2
        Band 2
            Band 3
            Band 3
```

This property is only set for a ProBindingSource bound to a ProDataSet with a recursive data-relation. A recursive data-relation can use a particular query in different bands. To access the correct query for the current row in this case, your application must use the BandIndex as a parameter for the data-relation's CURRENT-QUERY( ) method.

Note: The BandIndex is a 0-based index.

Data type: INTEGER

Access: PUBLIC Read-only

Applies to: Progress.Data.CancelCreateRowEventArgs class, Progress.Data.CreateRowEventArgs class

The following procedure is a CreateRow event handler designed to handle a ProDataSet with a recursive data-relation. First, it checks to see if the BandIndex is 0, which always uses the top query. If not, it uses the BandIndex as the parameter for the CURRENT-QUERY( ) method to find the handle of the correct instance of the data-relation query.

```
PROCEDURE recursiveRelationCreateRow:

DEFINE INPUT PARAMETER sender AS System.Object.
DEFINE INPUT PARAMETER args   AS Progress.Data.CreateRowEventArgs.

DEFINE VARIABLE hBuffer   AS HANDLE.
DEFINE VARIABLE hQuery    AS HANDLE.
DEFINE VARIABLE hTopQuery AS HANDLE.
DEFINE VARIABLE hRelation AS HANDLE.

hBuffer = args:BufferHdl.
IF args:BandIndex EQ 0 THEN
  hQuery = hTopQuery.
ELSE
  hQuery = hRelation:CURRENT-QUERY(args:BandIndex).

hBuffer:BUFFER-CREATE().
hQuery:CREATE-RESULT-LIST-ENTRY().
args:Created = TRUE.

END.
```

See also

BufferName property, CancelCreateRow event, Created property, CreateRow event

---

Batching property
(Windows only; GUI for .NET only)

Indicates whether record batching is enabled for the BindingSource object. Set to TRUE to enable record batching. Set to FALSE to disable record batching. The default value is FALSE.

You can use this property when the BindingSource object is bound to a query. If bound to a ProDataSet object, this property applies only to the top-level table displayed in the .NET control.

Data type: LOGICAL

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

When record batching is enabled and some user action in the bound .NET control retrieves the last record the current result set, the BindingSource object triggers an OffEnd event. You can use this event to retrieve the next batch of records.

See also

OffEnd event (.NET), RowsAdded property

BlockId property

The `BlockId` property identifies the `Block` VST. This VST contains information about a specific block. You can pass the `BlockId` property to the `GetVSTHandle()` method to return the handle to the `Block` VST. This property is read-only.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

Boolean property

Returned by the GetType method on a JsonObject or JsonArray indicating that the JSON data type for the specified property or element is boolean.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Json.ObjectModel.JsonDataType class

BufferHdl property
(Windows only; GUI for .NET only)

For a create row operation, this is the buffer handle for the newly created row. For a cancel create row operation, this is the buffer handle to the previously created row to delete.

You can use this property when the BindingSource object is bound to a ProDataSet object. This property applies to all tables displayed in the .NET control.

When the BindingSource object is bound to a query, this property returns the Unknown value (?).

Data type: HANDLE

Access: PUBLIC Read-only

Applies to: Progress.Data.CancelCreateRowEventArgs class, Progress.Data.CreateRowEventArgs class

See also

BufferName property, CancelCreateRow event, Created property, CreateRow event

BufferName property
(Windows only; GUI for .NET only)

For a create row operation, this is the buffer name for the newly created row. For a cancel create row operation, this is the buffer name of the previously created row to delete.

You can use this property when the BindingSource object is bound to a ProDataSet object. This property applies to all tables displayed in the .NET control.

When the BindingSource object is bound to a query, this property returns the Unknown value (?).

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Data.CancelCreateRowEventArgs class, Progress.Data.CreateRowEventArgs class

See also

BlockId property, CancelCreateRow event, Created property, CreateRow event

---

BuffStatusId property

The `BuffStatusId` property identifies the `BuffStatus` VST. This VST displays the status of buffers, such as the number of buffers that are in the buffer cache, that are currently in use, that are empty, or that are on the Least Recently Used (LRU) chain, page writer queue, or checkpoint queue. You can pass the `BuffStatusId` property to the `GetVSTHandle()` method to return the handle to the `BuffStatus` VST. This property is read-only.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

ByteOrder property

An internal indicator designating the byte-order of a MemoryOutputStream object's `Data` property. By default, the `ByteOrder` value is set to `PROGRESS.IO.ByteOrder:HostByteOrder`. Other possible values are `PROGRESS.IO.ByteOrder:LittleEndian` and `PROGRESS.IO.ByteOrder:BigEndian`.

See Progress.IO.ByteOrder enumeration for more information about these values.

Data type: Progress.IO.ByteOrder

Access: PUBLIC Read-only

Applies to: Progress.IO.MemoryOutputStream class

BytesWritten property

Total number bytes written to the current MemoryOutputStream object.

Data type: INT64

Access: PUBLIC Read-only

Applies to: Progress.IO.MemoryOutputStream class

CallStack property

Contains the ABL procedure call stack indicating where an ABL error or stop object was thrown.

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Lang.ProError class, Progress.Lang.Stop class, System.Exception class (OpenEdge-enhanced)

Returns the contents of the call stack as a string when an ABL error or stop object or a .NET Exception object is thrown. If the error or stop occurs from the user interface, the procedure containing the latest WAIT-FOR statement will be at the top of the call stack.

Note: The `StackTrace` property of a .NET exception object also provides a stack trace from the implementation of the .NET object being called (that is, the stack of managed routine names and offsets).

For an error only, if the ERROR-STACK-TRACE attribute of the SESSION handle is set to FALSE, this property returns the Unknown value (?). To enable the `CallStack` property, set the SESSION:ERROR-STACK-TRACE attribute to TRUE, or use the `-errorstack` startup parameter at session startup.

For a stop, the value of ERROR-STACK-TRACE is irrelevant and `CallStack` always returns content from the procedure call stack.

Caution: The ERROR-STACK-TRACE is set to FALSE by default because maintaining a call stack for error handling requires overhead that is not warranted in a production environment.

For more information, see ERROR-STACK-TRACE attribute reference entry. For more information on `-errorstack`, see *Startup Command and Parameter Reference*.

The call stack contains one entry for each item on the stack, starting with the last item, which is where the error or stop occurred. Each entry is delimited by the newline character. Each entry contains three values, separated by whitespace:

- Main block (`.p`), internal procedure, user-defined function, or method name
- Complete path of the file containing the user-defined function, internal procedure, or name of the method or property body
- Line number at which the error or stop occurred

In addition, if part of the call stack is from an OpenEdge application server's stack trace, `"Server StackTrace:"` appears above the server portion of the stack.

The following summarizes the syntax for a call stack entry, depending on the source.

Syntax

For a main block (`.p`):

```
file-name at line line-number (full-file-name)
```

For an internal procedure:

```
internal-procedure-name file-name at line line-number (full-file-name)
```

For a user-defined function:

```
user-defined-function-name file-name at line line-number (full-file-name)
```

For a class property or method:

```
property-or-method-name class-name at line line-number (full-file-name)
```

file-name

The name of the .p or .w file as it appears in the RUN statement which invoked the file.

line-number

The line number in the source code file where the error or stop was raised.

full-file-name

The *full-file-name* is identical to the *file-name*, but is prepended by the PROPATH element used by the RUN statement or class member invocation to identify the file.

internal-procedure-name

The name of an internal procedure in a .p or .w file.

user-defined-function-name

The name of a user-defined function in a .p or .w file.

property-or-method-name

The name of a property or method whose logic was invoked on *class-name*.

class-name

The name of the class, including the package name.

See also

ERROR-STACK-TRACE attribute, Progress.Lang.Error interface Progress.Lang.LockConflict class, Progress.Lang.StopAfter class, Progress.Lang.StopError class, Progress.Lang.UserInterrupt class

CanRead property

Returns TRUE if the property has a GET accessor. Otherwise, it returns FALSE.

Data type: LOGICAL

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Property class

CanWrite property

Returns TRUE if the property has a SET accessor. Otherwise, it returns FALSE.

Data type: LOGICAL

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Property class

ChildAllowEdit property
(Windows only; GUI for .NET only)

Indicates whether the .NET control should allow the user to edit values in the specified child temp-table buffer in the bound ABL data source object. The default value is TRUE.

Use this indexed property when the BindingSource object is bound to a ProDataSet object that contains child buffers.

Data type: LOGICAL

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

Syntax

```
ChildAllowEdit [ buffer-handle | buffer-name ]
```

buffer-handle

The handle to the child buffer to which this property applies.

buffer-name

The name of the child buffer to which this property applies.

Use this property only with .NET controls that support an edit operation.

See also

ActBufferId property

ChildAllowNew property
(Windows only; GUI for .NET only)

Indicates whether the .NET control should allow the user to add new records to the specified child temp-table buffer in the bound ABL data source object. The default value is TRUE.

Use this indexed property when the BindingSource object is bound to a ProDataSet object that contains child buffers.

Data type: LOGICAL

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

Syntax

```
ChildAllowNew[ buffer-handle | buffer-name ]
```

buffer-handle

The handle to the child buffer to which this property applies.

buffer-name

The name of the child buffer to which this property applies.

Use this property only with .NET controls that support an add operation.

See also

AllowNew property

ChildAllowRemove property
(Windows only; GUI for .NET only)

Indicates whether the .NET control should allow the user to remove records from the specified child temp-table buffer in the bound ABL data source object. The default value is TRUE.

Use this indexed property when the BindingSource object is bound to a ProDataSet object that contains child buffers.

Data type: LOGICAL

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

Syntax

```
ChildAllowRemove [ buffer-handle | buffer-name ]
```

buffer-handle

The handle to the child buffer to which this property applies.

buffer-name

The name of the child buffer to which this property applies.

Use this property only with .NET controls that support a remove operation.

See also

AllowRemove property

ChildInputValue property
(Windows only; GUI for .NET only)

Returns a Progress.Data.InputValue instance containing input values for all fields in the current row of the specified child temp-table displayed in the bound .NET control. Use the indexers in this instance to access the input value of a specific field in the row.

Use this property when the BindingSource object is bound to a ProDataSet object that contains child buffers.

Data type: Progress.Data.InputValue class

Access: PUBLIC Read-only

Applies to: Progress.Data.BindingSource class

Syntax

```
ChildInputValue[ buffer-handle | buffer-name ]
```

buffer-handle

A HANDLE variable that represents the handle to the child temp-table buffer whose input values you want to access.

buffer-name

A CHARACTER expression that evaluates to the name of the child temp-table buffer whose input values you want to access.

The following code fragment illustrates how to use this property:

```
DEFINE VARIABLE orderInpVals AS Progress.Data.InputValue NO-UNDO.
DEFINE VARIABLE orderNum     AS INTEGER                  NO-UNDO.

bufName = args:Row:Band:KEY.

IF bufName EQ "ttOrder" THEN DO:
  orderInpVals = myBindingSource:ChildInputValue[bufName].
  /* InputValue indexer is 0-based */
  orderNum = UNBOX( orderInpVals[0] ).
  IF orderNum > 10000 THEN DO:
    MESSAGE "You have reached max number of orders."
    ...
  END.
END.
```

See also

InputValue property, Progress.Data.InputValue class

ClientContextId property

Contains a client context identifier (CCID), which can identify a user login session and its associated identity and application context.

Data type: CHARACTER

Access: PUBLIC Readable/Writeable

Applies to: Progress.Lang.OERequestInfo class

Every application server client automatically generates a 22-character, Base64-encoded UUID value for this property as an initial value when the client creates the Server object handle for the application server (using the CREATE SERVER statement). This initial value is generated regardless of the operating mode of the application server and is used to set the `ClientContextId` property on the Progress.Lang.OERequestInfo instance referenced by the REQUEST-INFO attribute on the server object handle.

The client sends the value of this property to the application server when it connects to the application server using the CONNECT( ) method (Progress Application Server for OpenEdge) on the server handle and when it calls a remote external procedure using the RUN statement. The application server session can then retrieve this value by reading this property on the Progress.Lang.OERequestInfo instance referenced by the on the SESSION system handle.

You can also customize the behavior of this setting. In the:

- **Client session**
  - You can set a custom CCID value for this property referenced through the REQUEST-INFO attribute both **before** connecting the application server and **before** calling a remote external procedure.
  - You can terminate any further transmission of this property value to the application server by setting the Unknown value (?) for this property referenced through the REQUEST-INFO attribute **before** making any remote external procedure calls to the application server.
- **Application server session**
  - You can have the application server set a custom CCID value for this property on the Progress.Lang.OERequestInfo instance referenced through the CURRENT-RESPONSE-INFO attribute on the SESSION system handle **before** the end of the current application server request. Then the client returns the application server custom value to the application server when running subsequent remote requests.
  - You can terminate any further transmission of this property value to the application server from the client by setting the Unknown value (?) for this property referenced through the CURRENT-RESPONSE-INFO attribute on the SESSION system handle **before** the end of the current application server request.

When setting a custom CCID value for this property, you can use the following assignment to obtain a 22-character, Base64-encoded UUID value:

```
DEFINE VARIABLE ccid AS CHARACTER NO-UNDO.
ccid= SUBSTRING(BASE64-ENCODE (GENERATE-UUID), 1, 22).
```

However, you only need to use a non-blank character string with enough uniqueness to satisfy the requirements of your application architecture.

In whatever way an initial or new CCID value for this property is set, during each client request to the application server, the current value is automatically copied from the OERequestInfo instance referenced by the CURRENT-REQUEST-INFO attribute on the application server's SESSION handle to the instance reference by the CURRENT-RESPONSE-INFO attribute and returned to the client using the instance referenced by the RESPONSE-INFO attribute on the client's server object handle (or for an asynchronous request, on the asynchronous request object handle available in the asynchronous event procedure or method), which is then automatically copied to the instance referenced by the REQUEST-INFO attribute on the client's server object handle when the client makes its next remote request.

So, at any point in this round trip, a new value can be set through the application server's CURRENT-RESPONSE-INFO attribute before the request ends, or through the client's REQUEST-INFO attribute before the client's next remote request begins, and the new value continues its journey between client and application server and back, again, with each request.

Notes

- For each of the handle attributes that reference an instance of Progress.Lang.OERequestInfo, the attribute actually references this object as a Progress.Lang.Object. Therefore, you must cast the object reference down to a Progress.Lang.OERequestInfo in order to reference the object's OERequestInfo class properties.
- If you want an application server session running a client request to use the client's CCID value to connect and make remote requests as an application server client to yet another application server, you must assign the `ClientContextId` property available through the REQUEST-INFO attribute on the server object handle for the other application server to the value of the `ClientContextId` property available through the CURRENT-REQUEST-INFO attribute on the SESSION handle for the current application server session.
- This property works equally well for both the session-managed and session-free application server application models.

See also

CURRENT-REQUEST-INFO attribute, CURRENT-RESPONSE-INFO attribute, REQUEST-INFO attribute, RESPONSE-INFO attribute

---

Clear( ) method (Class)

Releases all information in a `Progress.Lang.ParameterList` object and sets the number of parameters to zero. Use of this method is effective for performance optimization by reusing a `Progress.Lang.ParameterList` object.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.ParameterList class

Syntax

```
Clear( )
```

See also

Invoke( ) method (Class), NumParameters property (ParameterList), SetParameter( ) method (Class)

Clear( ) method (MemoryOutputStream)

Causes the MemoryOutputStream object's `Data` and `Length` properties to be freed and their size set to their initial values when the MemoryOutputStream object was instantiated.

Return type: VOID

Access: PUBLIC

Applies to: Progress.IO.MemoryOutputStream class

Syntax

```
clear ( )
```

Notes

- The `Clear( )` method does not reset a stream's byte-order if the byte-order was changed by the `SetByteOrder( )` method.
- The following table illustrates values that result from constructors and `Write( )`/`Clear( )` methods:

Method/Constructor	Values	
	Obj:Length	Obj:BytesWritten
`Obj = NEW MemoryOutputStream()`	0	0
`Obj:Write("12345")`	5	5
`Obj:Clear()`	0	0
`Obj = NEW MemoryOutputStream(100)`	100	0
`Obj:Write("12345678")`	100	8
`Obj:Clear()`	100	0

Clone( ) method (JsonArray)

Creates a copy of a JsonArray instance and returns an object reference to the new instance.

Return type: Progress.Lang.Object class

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
Clone( )
```

If any object references within this object instance are invalid, a JsonError is raised.

Clone( ) method (JsonObject)

Creates a copy of a JsonObject instance and returns an object reference to the new instance.

Return type: Progress.Lang.Object class

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
Clone( )
```

If any object references within this object instance are invalid, a JsonError is raised.

Clone( ) method (Progress.Lang.Object)

Creates a copy of an object instance and returns an object reference of the copy.

Return type: Progress.Lang.Object class

Access: PUBLIC

Applies to: Progress.Lang.Object class

Syntax

```
Clone( )
```

This method has no default behavior. You must override this method in a user-defined class. If you invoke this method without overriding it, ABL generates an error message and returns the Unknown value (?).

Close( ) method (IO)

Closes the stream and sets the `Closed` property of the stream object to TRUE.

Return type: VOID

Access: PUBLIC

Applies to: Progress.IO.InputStream class, Progress.IO.OutputStream class

Syntax

```
Close( )
```

Notes

- If you override this method to perform additional clean up tasks, you should set the `Closed` property to TRUE or call `SUPER:Close( )` to get the default behavior.

See also

Closed property

Closed property

Indicates whether the stream is closed. The initial value is FALSE. Invoking the `Close( )` method on a stream sets this property to TRUE.

Data type: LOGICAL

Access: PUBLIC Read-only

Applies to: Progress.IO.InputStream class, Progress.IO.OutputStream class

CompareTo( ) method

Compares the enum instance to another specified enum instance based on their underlying numeric values. The method returns -1, 0, or 1 depending on whether the numerical value of the enum instance is less than, equal to, or greater than the specified enum instance, respectively. If the specified enum instance is not initialized, the method returns the Unknown value (?).

Return type: INTEGER

Access: PUBLIC

Applies to: Progress.Lang.Enum class

Syntax

```
CompareTo ( INPUT otherEnum AS Progress.Lang.Enum )
```

otherEnum

A reference to a `Progress.Lang.Enum` instance. The enum type of this instance must be the same as the enum type of the instance the method is invoked on.

The following example compares two enum instances of type `Days`:

```
ENUM Days:
     DEFINE ENUM Monday
                 Tuesday
                 Wednesday
                 Thursday
                 Friday
                 Saturday
                 Sunday.
END ENUM.
```

```
DEFINE VARIABLE myDeliveryDay AS Days NO-UNDO.

myDeliveryDay = Days:Sunday.

IF myDeliveryDay:CompareTo(Days:Friday) = 1 THEN
  MESSAGE "That is not a valid delivery day.".
```

Notes

- The method raises error if the two enum instances are not the same type.
- You can also use the >, <, =, <>, >=, and <= operators to perform the same type of comparison. However, the enum type must be known at compile time for use with those operators. You can use this method if you need more flexibility, because this method can compare references of type `Progress.Lang.Enum`.

See also

ENUM statement

Complement( ) method
(Windows only; GUI for .NET only)

Performs a bitwise complement (NOT) operation on the underlying value of the specified enumeration type and returns a new instance. You typically use this method to unset (turn off) a flag.

Return type: System.Enum class (from the .NET Framework)

Access: PUBLIC STATIC

Applies to: Progress.Util.EnumHelper class

Syntax

```
EnumHelper:Complement ( INPUT enum AS CLASS System.Enum )
```

enum

Static reference to an enumeration value, or a reference to an enumeration type instance.

Specify a static reference to an enumeration value using the following syntax:

```
enum-type:enum-member
```

Where *enum-type* is the type name of the enumeration and *enum-member* is a member name of the enumeration. For example:

```
System.Windows.Forms.AnchorStyles:Bottom
```

The following example uses this method to invert flags:

```
USING System.Windows.Forms.* FROM ASSEMBLY.
USING Progress.Util.* FROM ASSEMBLY.

DEFINE VARIABLE rMyStyle AS System.Enum NO-UNDO.

/* Adds Bottom and Right to the list */
rMyStyle = EnumHelper:Or( AnchorStyles:Bottom, AnchorStyles:Right ).
DISPLAY STRING( rMyStyle ) FORMAT "x(20)" SKIP.
/* Displays "Bottom, Right" */

/* Sets all flags that are not set and clears all that are */
rMyStyle = EnumHelper:Complement( rMyStyle ).
DISPLAY STRING( rMyStyle ) FORMAT "x(20)" SKIP.
/* Displays "-11", which includes Top and Left but not Bottom or Right */

rMyStyle = EnumHelper:Complement( rMyStyle ).
DISPLAY STRING( rMyStyle ) FORMAT "x(20)".
/* Displays "Bottom, Right" */
```

If you want to use the returned System.Enum instance as a different enumeration type, you must cast the instance to that specific type using the CAST function.

For information on the System.Enum class and .NET enumeration types, refer to the .NET Framework class library.

See also

CAST function

Count property
(Windows only; GUI for .NET only)

The number of records in the result set for the query associated with the top-level table displayed in the bound .NET control.

You can use this property when the BindingSource object is bound to a query or a buffer. If bound to a ProDataSet object, this property applies only to the top-level query. When bound to a buffer, the value is always 1.

Data type: INTEGER

Access: PUBLIC Read-only

Applies to: Progress.Data.BindingSource class

Note: For queries with large result sets, you might want to set the MaxDataGuess property to provide bound .NET controls with a value to use for initialization. Otherwise, the ProBindingSource must read every record in the result set to provide a value for the Count property for the control to use.

See also

MaxDataGuess property

Create( ) method (CHARACTER)
(Windows only; GUI for .NET only)

Use this set of Create( ) method overloads to initialize CHARACTER values in ABL.

Note: This method is used primarily by the Visual Designer during code generation. You should not expect to use this method directly.

Return type: CHARACTER

Access: PUBLIC STATIC

Applies to: Progress.Util.StringHelper class

Syntax

```
StringHelper:Create ( method-parameters )
```

method-parameters

A CHARACTER expression that contains one or more parameters for this method (based on the method overload).

The following table describes the Create( ) method overloads for initializing CHARACTER values.

Table 134: Create( ) method overloads for CHARACTER values

Method overload syntax	Description
`Create( INPUT chars AS CHARACTER EXTENT )`	Returns a CHARACTER initialized to the value indicated by an array of characters
`Create( INPUT chars AS CHARACTER,` `INPUT count AS INTEGER )`	Returns a CHARACTER initialized to the value indicated by a specified character repeated a specified number of times

`Create( INPUT chars AS CHARACTER EXTENT,`
`INPUT startIndex AS INTEGER,`
`INPUT length AS INTEGER )`
	Returns a CHARACTER initialized to the value indicated by a specified array of characters, a starting character position within that array, and a length

The following example uses this method to initialize a CHARACTER variable with the string "AAAAAAA", and then displays the result:

```
USING Progress.Util.* FROM ASSEMBLY.

DEFINE VARIABLE aString AS CHARACTER NO-UNDO.

aString = StringHelper:Create( "A", 7 ).

Display aString.
```

See also

Create( ) method (DATETIME), Create( ) method (DECIMAL)

Create( ) method (DATETIME)
(Windows only; GUI for .NET only)

Use this set of Create( ) method overloads to initialize DATETIME values in ABL.

Note: This method is used primarily by the Visual Designer during code generation. You should not expect to use this method directly.

Return type: DATETIME

Access: PUBLIC STATIC

Applies to: Progress.Util.DateTimeHelper class

Syntax

```
DateTimeHelper:Create ( method-parameters )
```

method-parameters

A CHARACTER expression that contains one or more parameters for this method (based on the method overload).

The following table describes the Create( ) method overloads for initializing DATETIME values.

Table 135: Create( ) method overloads for DATETIME values

Method overload syntax	Description
`Create( INPUT ticks AS INT64 )`	Returns a DATETIME initialized to a specified number of ticks (time values measured in 100-nanosecond units)
`Create( INPUT ticks AS INT64,` `  INPUT kind AS System.DateTimeKind )`	Returns a DATETIME initialized to a specified number of ticks and to Coordinated Universal Time (UTC) or local time

`Create( INPUT year AS INTEGER,`
`  INPUT month AS INTEGER,`
`  INPUT day AS INTEGER )`
	Returns a DATETIME initialized to the specified year, month, and day

`Create( INPUT year AS INTEGER,`
`  INPUT month AS INTEGER,`
`  INPUT day AS INTEGER,`
`  INPUT calendar AS`
`  System.Globalization.Calendar )`
	Returns a DATETIME initialized to the specified year, month, and day for the specified calendar

`Create( INPUT year AS INTEGER,`
`  INPUT month AS INTEGER,`
`  INPUT day AS INTEGER,`
`  INPUT hour AS INTEGER,`
`  INPUT minute AS INTEGER,`
`  INPUT second AS INTEGER )`
	Returns a DATETIME initialized to the specified year, month, day, hour, minute, and second

`Create( INPUT year AS INTEGER,`
`  INPUT month AS INTEGER,`
`  INPUT day AS INTEGER,`
`  INPUT hour AS INTEGER,`
`  INPUT minute AS INTEGER,`
`  INPUT second AS INTEGER,`
`  INPUT calendar AS`
`  System.Globalization.Calendar )`
	Returns a DATETIME initialized to the specified year, month, day, hour, minute, and second for the specified calendar

`Create( INPUT year AS INTEGER,`
`  INPUT month AS INTEGER,`
`  INPUT day AS INTEGER,`
`  INPUT hour AS INTEGER,`
`  INPUT minute AS INTEGER,`
`  INPUT second AS INTEGER,`
`  INPUT kind AS System.DateTimeKind )`
	Returns a DATETIME initialized to the specified year, month, day, hour, minute, and second and to Coordinated Universal Time (UTC) or local time

Method overload syntax	Description

`Create( INPUT year AS INTEGER,`
`INPUT month AS INTEGER,`
`INPUT day AS INTEGER,`
`INPUT hour AS INTEGER,`
`INPUT minute AS INTEGER,`
`INPUT second AS INTEGER,`
`INPUT millisecond AS INTEGER )`
	Returns a DATETIME initialized to the specified year, month, day, hour, minute, second, and millisecond

`Create( INPUT year AS INTEGER,`
`INPUT month AS INTEGER,`
`INPUT day AS INTEGER,`
`INPUT hour AS INTEGER,`
`INPUT minute AS INTEGER,`
`INPUT second AS INTEGER,`
`INPUT millisecond AS INTEGER,`
`INPUT calendar AS`
`System.Globalization.Calendar )`
	Returns a DATETIME initialized to the specified year, month, day, hour, minute, second, and millisecond for the specified calendar

`Create( INPUT year AS INTEGER,`
`INPUT month AS INTEGER,`
`INPUT day AS INTEGER,`
`INPUT hour AS INTEGER,`
`INPUT minute AS INTEGER,`
`INPUT second AS INTEGER,`
`INPUT millisecond AS INTEGER,`
`INPUT kind AS System.DateTimeKind )`
	Returns a DATETIME initialized to the specified year, month, day, hour, minute, second, and millisecond and to Coordinated Universal Time (UTC) or local time

`Create( INPUT year AS INTEGER,`
`INPUT month AS INTEGER,`
`INPUT day AS INTEGER,`
`INPUT hour AS INTEGER,`
`INPUT minute AS INTEGER,`
`INPUT second AS INTEGER,`
`INPUT millisecond AS INTEGER,`
`INPUT calendar AS`
`System.Globalization.Calendar,`
`INPUT kind AS System.DateTimeKind )`
	Returns a DATETIME initialized to the specified year, month, day, hour, minute, second, and millisecond for the specified calendar and to Coordinated Universal Time (UTC) or local time

The following example uses this method to initialize a DATETIME variable with the date, July 28, 1986, and then displays the result:

```
USING Progress.Util.* FROM ASSEMBLY.

DEFINE VARIABLE aDateTime AS DATETIME NO-UNDO.

aDateTime = DateTimeHelper:Create( 1986, 7, 28 ).

Display aDateTime.
```

See also

Create( ) method (CHARACTER), Create( ) method (DECIMAL)

Create( ) method (DECIMAL)
(Windows only; GUI for .NET only)

Use this set of Create( ) method overloads to initialize DECIMAL values in ABL.

Note: This method is used primarily by the Visual Designer during code generation. You should not expect to use this method directly.

Return type: DECIMAL

Access: PUBLIC STATIC

Applies to: Progress.Util.DecimalHelper class

Syntax

```
DecimalHelper:Create ( method-parameters )
```

method-parameters

A CHARACTER expression that contains one or more parameters for this method (based on the method overload).

The following table describes the Create( ) method overloads for initializing DECIMAL values.

Table 136: Create( ) method overloads for DECIMAL values

Method overload syntax	Description
`Create( INPUT value AS DECIMAL (DOUBLE) )`	Returns a DECIMAL initialized to the value of the specified double-precision floating-point number
`Create( INPUT value AS INTEGER )`	Returns a DECIMAL initialized to the value of the specified integer value
`Create( INPUT value AS INTEGER EXTENT 4 )`	Returns a DECIMAL initialized to a decimal value represented in binary and contained in a specified array
`Create( INPUT value AS INT64 )`	Returns a DECIMAL initialized to the value of the specified integer value
`Create( INPUT value AS DECIMAL (FLOAT) )`	Returns a DECIMAL initialized to the value of the specified single-precision floating-point number
`Create( INPUT value AS INT64 (UNSIGNED-INTEGER) )`	Returns a DECIMAL initialized to the value of the specified unsigned integer value
`Create( INPUT value AS DECIMAL (UNSIGNED-INT64) )`	Returns a DECIMAL initialized to the value of the specified unsigned integer value
`Create( INPUT lo AS INTEGER, INPUT mid AS INTEGER, INPUT hi AS INTEGER, INPUT isNegative AS LOGICAL, INPUT scale AS INTEGER (UNSIGNED-BYTE) )`	Returns a DECIMAL initialized from parameters specifying its constituent parts

The following example uses this method to initialize a DECIMAL variable with the value 23.45, and then displays the result:

```
USING Progress.Util.* FROM ASSEMBLY.

DEFINE VARIABLE aDecimal AS DECIMAL NO-UNDO.

aDecimal = DecimalHelper:Create( 2345, 0, 0, FALSE, 2 ).

Display aDecimal.
```

See also

Create( ) method (CHARACTER), Create( ) method (DATETIME)

Created property
(Windows only; GUI for .NET only)

Use this property in response to a CreateRow event to indicate whether the record was successfully created in the bound ABL data source object. Set to TRUE if the record was successfully created. Set to FALSE if the record was not created. The default value is TRUE.

Data type: LOGICAL

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.CreateRowEventArgs class

If the record was successfully created in the bound ABL data source object, you must also add the record to the query's result set using the CREATE-RESULT-LIST-ENTRY( ) method. Otherwise, the data in the control will not be synchronized with its data source.

Caution: After creating a row, do not reopen the query or invoke the Refresh( ) method as this might produce unexpected results.

See also

BlockId property, BufferName property

Custom property

An authentication status code that indicates a custom error condition that is defined and returned by an ABL callback procedure.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

Data property

The MEMPTR holding memory stream data.

Data type: MEMPTR

Access: PUBLIC Read-only

Applies to: Progress.IO.MemoryOutputStream class, Progress.IO.MemoryInputStream class

Notes

- Any attempt to use memory-writing statements or functions other than `MemoryOutputStream` or `MemoryInputStream` methods causes the AVM to generate a runtime error message indicating that the MEMPTR is read-only.

- You can duplicate the `Data` property by assigning it to a MEMPTR variable.

DataType property

Returns an instance of `Progress.Reflect.DataType` that indicates the data type (e.g., `Character`, `Integer`, etc.) of the given data member, property, or parameter. If the data member, property, or parameter is a class-based object, then the `DataTypeName` property returns a class type name, including package, as a CHARACTER expression.

Data type: Progress.Reflect.DataType enumeration

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Parameter class, Progress.Reflect.Variable class

See also

DataTypeName property (Reflect)

DataTypeName property (Reflect)

When the associated `DataType` property indicates a class-based object, `DataTypeName` returns a class type name, including package, as a CHARACTER expression. If `DataType` does not indicate an object, `DataTypeName` returns the Unknown value (?).

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Parameter class, Progress.Reflect.Variable class

Note: For .NET class types, `DataTypeName` returns the class type name used in the ABL, not the assembly-qualified name.

See also

DataType property

DbStatusId property

The `DbStatusId` property identifies the `DbStatus` VST. This VST displays the status of a temp-table database in use. You can pass the `DbStatusId` property to the `GetVSTHandle()` method to return the handle to the `DbStatus` VST. This property is read-only.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

DeclaringClass property

Returns a `Progress.Lang.Class` instance representing the class in which the constructor, event, method, data member, or property is defined.

Data type: Progress.Lang.Class class

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Event class, Progress.Reflect.Method class, Progress.Reflect.Variable class

When applied to a class member (method, property, or event) that is overridden, the `DeclaringClass` property returns the most-derived class containing the implementation of the member.

For example, consider the classes defined in the following code fragment:

```
CLASS ClassA:
  METHOD PUBLIC VOID MethodA:
    MESSAGE "A method has been called." VIEW-AS ALERT-BOX.
  END METHOD.
END CLASS.

CLASS ClassB INHERITS ClassA:
END CLASS.

CLASS ClassC INHERITS ClassB:
  METHOD PUBLIC OVERRIDE VOID MethodA:
    MESSAGE "An overriden method has been called." VIEW-AS ALERT-BOX.
  END METHOD.
END CLASS.
```

Calling `GetMethod("MethodA", params)` on instances of `Progress.Lang.Class` derived from each of the three classes would return `Progress.Reflect.Method` instances with the `DeclaringClass` properties set as shown in the table:

`Progress.Lang.Class` instance	**Class indicated by the `DeclaringClass` property**
`ClassA`	`ClassA`
`ClassB`	`ClassA`
`ClassC`	`ClassC`

See also: OriginatingClass property

Deserialize( ) method (BinarySerializer)

Deserializes an ABL class-based object from the specified binary stream.

Return type: Progress.Lang.Object class

Access: PUBLIC

Applies to: Progress.IO.BinarySerializer class

Syntax

```
Deserialize( INPUT input-stream AS Progress.IO.InputStream )
```

input-stream

The stream from which the class-based object is deserialized.

The following code sample serializes and deserializes `myObj`:

```
DEFINE VARIABLE myFileOutStream AS Progress.IO.FileOutputStream.
DEFINE VARIABLE myFileInStream  AS Progress.IO.FileInputStream.
DEFINE VARIABLE mySerializer    AS Progress.IO.BinarySerializer.
DEFINE VARIABLE myObj           AS Acme.MyClass.

myObj = NEW  Acme.MyClass().

mySerializer = NEW Progress.IO.BinarySerializer().

/* Serialize object */
myFileOutStream = NEW Progress.IO.FileOutputStream("MyClass.bin").
mySerializer:Serialize(myObj, myFileOutStream).
myFileOutStream:Close().

/* Deserialize object */
myFileInStream = NEW Progress.IO.FileInputStream("MyClass.bin").
myObj = CAST(mySerializer:Deserialize(myFileInStream), Acme.MyClass).
myFileInStream:Close().
```

Notes

- If a class member is not present in the input stream, then that member will be assigned the default value for its type.
- If the input stream is unknown or cannot be read from, the AVM raises an error.
- If the class of the object being deserialized changed between serialization and deserialization in a way that causes the object and its class to no longer match, the AVM raises an error.
- The current position in the stream must be at the beginning of the data for the serialized object for the deserialization to complete successfully.

See also

Serialize( ) method (BinarySerializer)

Deserialize( ) method (JsonSerializer)

Deserializes an ABL class-based object from the specified JSON stream.

Return type: Progress.Lang.Object class

Access: PUBLIC

Applies to: Progress.IO.JsonSerializer class

Syntax

```
Deserialize( INPUT input-stream AS Progress.IO.InputStream )
```

input-stream

The stream from which the class-based object is deserialized.

The following code sample deserializes the object in `myFileStream` to the object reference `myObj`:

```
DEFINE VARIABLE myFileOutStream AS Progress.IO.FileOutputStream.
DEFINE VARIABLE myFileInStream  AS Progress.IO.FileInputStream.
DEFINE VARIABLE mySerializer    AS Progress.IO.JsonSerializer.
DEFINE VARIABLE myObj           AS Acme.MyClass.

myObj = NEW  Acme.MyClass().

mySerializer = NEW Progress.IO.JsonSerializer(FALSE).

/* Serialize object */
myFileOutStream = NEW Progress.IO.FileOutputStream("MyClass.json").
mySerializer:Serialize(myObj, myFileOutStream).
myFileOutStream:Close().

/* Deserialize object */
myFileInStream = NEW Progress.IO.FileInputStream("MyClass.json").
myObj = CAST(mySerializer:Deserialize(myFileInStream), Acme.MyClass).
myFileInStream:Close().
```

Notes

- If a class member is not present in the input stream (for example, a private or protected member that was not defined as SERIALIZABLE), then that member is assigned its initial value.
- If the input stream is unknown or cannot be read from, the AVM raises an error.
- If the class of the object being deserialized changed between serialization and deserialization in a way that causes the object and its class to no longer match, the AVM raises an error.
- The current position in the stream must be at the beginning of the data for the serialized object for the deserialization to complete successfully.

See also

Serialize( ) method (JsonSerializer)

Device property

The name of the device on which the other AVM process is running that is holding the lock and thus causing the specified lock conflict.

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Lang.LockConflict class

Alternatively, this can be `"Dictionary"` if another process is doing schema updates through the Data Dictionary. The format of this name depends on the operating system, but it is the same name that currently displays in the **Lock Conflict** dialog box.

See also

CallStack property, TableName property, User property

Dispose( ) method
(Windows only; GUI for .NET only)

Cleans up resources associated with a BindingSource object before .NET releases it from memory.

Return type: VOID

Access: PUBLIC

Applies to: Progress.Data.BindingSource class

Syntax

```
Dispose( )
```

Call this method before you delete the object reference to a BindingSource instance.

Use this method especially when a BindingSource has a ProDataSet as its data source. For each row that is expanded in a hierarchical grid for a ProDataSet, the BindingSource creates a separate query. This can potentially consume a lot of memory. The ABL Virtual Machine (AVM) automatically deletes these queries when you delete the ProDataSet. However, if you want to maintain the ProDataSet beyond the scope of the BindingSource, calling the Dispose( ) method frees all of the memory for these queries without your having to delete the ProDataSet that is associated with them.

DisposeDialogOnClose property
(Windows only; GUI for .NET only)

For dialog boxes that inherit from `Progress.Windows.Form`, this property indicates whether or not the AVM automatically calls `Dispose( )` on the form when the dialog is closed. The default value is TRUE.

Data type: LOGICAL

Access: PUBLIC Readable/Writeable

Applies to: Progress.Windows.Form class

.NET itself does not call `Dispose( )` on forms shown as dialog boxes when they are closed, but when this property is true, the AVM automatically calls `Dispose( )`. The only reason to set this property to FALSE is if your application needs to re-use the form at a later time. You can change the default for all forms by using the startup parameter `-nodialogdispose`. See the *Startup Command and Parameter Reference* for more information.

EmbeddedWindow property
(Windows only; GUI for .NET only)

The handle to an ABL window whose client area is embedded in the client area of the specified .NET control container. This control container can be either a Progress.Windows.MdiChildForm or a Progress.Windows.WindowContainer (which you can add to any Progress.Windows.Form). After a .NET form with this embedded client area is displayed, you can interact with its ABL widgets using the supported widget attributes, methods, and events as if they were displayed in the original ABL window.

Data type: HANDLE

Access: PUBLIC Readable/Writeable

Applies to: Progress.Windows.MDIChildForm class, Progress.Windows.WindowContainer class

To reset a previously set instance of this property on a control container:

- The current ABL window referenced by this property must not yet be realized.
- If the current ABL window referenced by this property is realized, you must first delete this window before resetting the property to reference another ABL window.

Note that when you embed an ABL window in a .NET form, only the client area of the window is embedded. ABL ignores all other components of the ABL window, including border controls, menu bar, message area, and status area. For more information, see the reference entries for the Progress.Windows.MDIChildForm class and Progress.Windows.WindowContainer class.

When you embed the client area of an ABL window in a supported .NET control container, the various attributes, methods, and events of the embedded ABL window either function normally, change behavior, or have no function (are ignored), depending on the element. The following tables describe how these elements are affected by embedding the ABL window. For more information on the attributes, methods, and events of an ABL window, see the WINDOW widget reference entry.

Important: You must handle the `FormClosing` event on the main .NET form in order to delete the embedded ABL window when the form closes.

Note: You can use enhanced logging to trace the occurrence of attributes and methods in your application that are ignored when you access them on an embedded ABL window. For more information, see the IgnoredOps entry type described in the LOG-ENTRY-TYPES attribute reference entry.

The following table lists those attributes that work on an embedded ABL window in the same way as on a non-embedded window.

Table 137: Attributes unchanged on an embedded ABL window

BGCOLOR	CONTEXT-HELP-FILE[90]	DROP-TARGET
DYNAMIC	FIRST-CHILD	HANDLE
HWND	INSTANTIATING-PROCEDURE	KEEP-FRAME-Z-ORDER

[90] You can specify a context help file for all frames within an embedded window by setting its CONTEXT- HELP-FILE attribute. However, ABL does not support the CONTEXT-HELP attribute for embedded windows (see Table 139: Attributes ignored on an embedded ABL window). Instead, you must set the HelpButton property on the containing form in order to enable the context-help button on the form's caption bar. This containing form can be an MDI parent form, an MDI child form, or a form that contains a WindowContainer object. Note that MDI applications generally do not use the context-help feature on their forms; this feature is most commonly used in dialog boxes.

LAST-CHILD	MENU-KEY[91]	MENU-MOUSE[91]
MOUSE-POINTER	NAME	NEXT-SIBLING
NUM-DROPPED-FILES	NUM-SELECTED-WIDGETS[91]	POPUP-MENU[91]
PREV-SIBLING	PRIVATE-DATA	SCROLL-BARS[92]
THREE-D	TYPE	WINDOW

The following table describes attributes whose behavior on an embedded ABL window differs from how they behave on a non-embedded window.

Table 138: Attributes modified on an embedded ABL window

Attributes	Behavior changes
PARENT	This attribute specifies the parent of a window in an ABL window hierarchy. An embedded window cannot participate in this hierarchy. So, the PARENT attribute always returns the Unknown value (?) for an embedded window. ABL ignores any setting of this attribute on an embedded window.
SENSITIVE HIDDEN VISIBLE	The SENSITIVE attribute indicates if a widget is enabled to receive keyboard and mouse input. The default value for the SENSITIVE attribute on a window is TRUE. Setting this attribute to FALSE on an embedded window disables input to the embedded frames, but it does not disable the non-client portions of an embedding MDI child form or any form that contains a Progress.Windows.WindowContainer object. Similarly, setting VISIBLE to FALSE or HIDDEN to TRUE on an embedded window hides the embedded frames but does not hide the .NET control container into which they have been embedded.
X Y ROW COLUMN	When queried on an embedded window, the X, Y, ROW, and COLUMN attributes return 0, 0, 1, and 1, respectively, because these attributes return position values relative to the embedded window's container. The embedded window's position is fixed at the upper-left corner of the container's client area. ABL ignores any setting of these attributes on an embedded window. You must set the position of any Progress.Windows.WindowContainer within its containing form using appropriate .NET properties.

[91] These attributes work on an embedded window because they apply to actions performed on the frames and field-level widgets within the window.

[92] If you set the SCROLL-BARS attribute to TRUE on an embedded window, ABL handles the scrolling of frames within that window. The value of the AutoScroll property of the embedding form or WindowContainer has no effect on the scrolling features of these frames.

Attributes	Behavior changes
WIDTH-CHARS WIDTH-PIXELS HEIGHT-CHARS HEIGHT-PIXELS	The WIDTH-CHARS, WIDTH-PIXELS, HEIGHT-CHARS, and HEIGHT-PIXELS window attributes return the width and height, respectively, of the client area of the embedded window, just as they do in a non-embedded window. ABL ignores any setting of these attributes on an embedded window. You must set the size of any Progress.Windows.WindowContainer within its containing form using appropriate .NET properties. **Note:** The FULL-WIDTH-* and FULL-HEIGHT-* attributes include the size of the non-client areas of a window and, as such, are ignored for embedded windows (see Table 139: Attributes ignored on an embedded ABL window).
VIRTUAL-HEIGHT-CHARS VIRTUAL-HEIGHT-PIXELS VIRTUAL-WIDTH-CHARS VIRTUAL-WIDTH-PIXELS	These attributes work for embedded windows. They determine the size of the virtual window for scrolling frames in windows. As noted for the SCROLL-BARS attribute in Table 137: Attributes unchanged on an embedded ABL window, ABL handles the scrolling of frames within an embedded window. For embedded windows, ABL maintains a minimum size for the virtual window; you cannot make the size of this virtual window smaller than the physical size of the embedded window container because that would cause the container to become smaller.

The following table lists attributes on an embedded ABL window that ABL ignores. You can read or write these attributes without error, but they have no effect because their function has no meaning for the client area of an ABL window embedded in a .NET form. For the corresponding behavior in a .NET form object, use the appropriate .NET class members.

Table 139: Attributes ignored on an embedded ABL window

ALWAYS-ON-TOP	CONTEXT-HELP	CONTROL-BOX
DCOLOR	FGCOLOR[93]	FONT[93]
FULL-HEIGHT-CHARS[94]	FULL-HEIGHT-PIXELS[94]	FULL-WIDTH-CHARS[94]
FULL-WIDTH-PIXELS[94]	ICON	MAX-BUTTON
MAX-HEIGHT-CHARS[95]	MAX-HEIGHT-PIXELS[95]	MAX-WIDTH-CHARS[95]
MAX-WIDTH-PIXELS[95]	MENU-BAR	MESSAGE-AREA
MESSAGE-AREA-FONT	MIN-BUTTON	MIN-HEIGHT-CHARS[96]

[93] These attributes only affect a window's message area and are not inherited by frames placed inside the window. Thus, they will have no affect when the frames are removed from a window and placed inside a form.

[94] These attributes always return the Unknown value (?) when queried on an embedded window because there is no concept of "full" window size for an embedded window.

[95] These attributes always return the Unknown value (?) when queried on an embedded window because there is no concept of "maximum" window size for an embedded window.

[96] These attributes always return zero (0) when queried on an embedded window. An embedded window's size is determined by the size of its container, and that container's size is independent of the values of these ABL attributes.

MIN-HEIGHT-PIXELS[96]	MIN-WIDTH-CHARS[96]	MIN-WIDTH-PIXELS[96]
PFCOLOR	RESIZE	SCREEN-LINES
SHOW-IN-TASKBAR	SMALL-ICON	SMALL-TITLE
STATUS-AREA	STATUS-AREA-FONT	TITLE
TOP-ONLY	WINDOW-STATE	–

The following table describes how methods are supported on an embedded ABL window.

Table 140: Method support on an embedded ABL window

Methods	Support provided by ABL
END-FILE-DROP( ) GET-DROPPED-FILE( )	These methods work for file drag-and-drop operations on an embedded window.
GET-SELECTED-WIDGET( )	This method returns the handle of the selected frame in an embedded window.
LOAD-MOUSE-POINTER( )	This method specifies the mouse pointer to display when the pointer is moved over the embedded frames of the window. When a window is embedded in a form, the mouse pointer changes when the pointer is over these embedded frames, but it does not change when the pointer is over the non-client areas of the form. If you want the non-client areas of the form to have the same pointer as the embedded window, you must set the form pointer separately using the Cursor property on the form.
LOAD-ICON( ) LOAD-SMALL-ICON( ) MOVE-TO-BOTTOM( ) MOVE-TO-TOP( )	ABL ignores execution of these methods on an embedded window.

For an embedded ABL window, ABL frame-level (ENTRY, LEAVE, etc.) and field-level events (mouse button clicks, etc.) fire normally on the embedded frames and the widgets they contain (see the Handle-based Object Events Reference). However, most window-level events do not fire because the ABL window is not visualized. You must handle .NET form events in ABL using .NET event handlers (see the Class Events Reference). The following table describes how ABL supports window-level events for embedded ABL windows.

Table 141: Event support on an embedded ABL window

Events	Support provided by ABL
DROP-FILE-NOTIFY	This event fires when a file is dragged and dropped onto the client area of an embedded window whose DROP-TARGET attribute is set to TRUE.

Events	Support provided by ABL
WINDOW-RESIZED	This event fires whenever the embedded window container is resized, as follows: • A Progress.Windows.MdiChildForm can be resized either programmatically or by the user interacting with the form on screen. • A Progress.Windows.WindowContainer can be resized only programmatically, typically in response to handling a .NET form Resize event. By handling the ABL WINDOW-RESIZED event in a trigger, you can resize the embedded frames and their contents to fit the new size of their container.
ENTRY LEAVE PARENT-WINDOW-CLOSE WINDOW-CLOSE WINDOW-MAXIMIZED[97] WINDOW-MINIMIZED WINDOW-RESTORED[97]	The AVM never generates these events for an embedded window.

END_OF_ARRAY property

A symbolic constant used in length arguments that indicates the end of the array.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Equals( ) method

Compares the object reference for the current class instance to the specified object reference. If the object references (regardless of type) point to the same object instance, this method returns TRUE. Otherwise, it returns FALSE. When applied to enums, the method compares the object reference for the current instance to the specified enum instance. If the object reference points to an instance of the same enum type and the same underlying numeric value, this method returns TRUE. If the object reference and the specified enum instance are of the same enum type but have different underlying numeric values, it returns FALSE. If the object reference and enum instance are of different enum types or the object reference is to a class instance, the method raises an error.

[97] ABL does not map the WINDOW-MAXIMIZED and WINDOW-RESTORED events from the corresponding .NET form events. However, for a Progress.Windows.MdiChildForm, the WINDOW-RESIZED event fires on the embedded window when the child form is maximized or restored.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.Object class, System.Object class (from the .NET Framework)

Syntax

```
Equals ( INPUT OtherObj AS Progress.Lang.Object )
```

OtherObj

The other object reference with which to compare.

Extent property

When the given data member, parameter, or property is an array, `Extent` returns the size (extent) of the array. If the array is indeterminate in its initial state with no size, `Extent` returns the Unknown value (?). If the data member, parameter, or property is not an array, `Extent` returns 0.

Data type: INTEGER

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Parameter class, Progress.Reflect.Variable class

---

FieldIndex property
(Windows only; GUI for .NET only)

Use this property in response to a SortRequest event to indicate the 1-based index position of the field on which to sort the records in the ABL data source object.

Data type: INTEGER

Access: PUBLIC Read-only

Applies to: Progress.Data.SortRequestEventArgs class

This index is based on the order of the fields as specified in the Progress.Data.BindingSource class constructor. If you specify an asterisk ("*") in the constructor, the index is based on the order of the fields in the database record. If the field on which you are sorting is an array field, the ArrayIndex property is set to the 1-based index position of the array element on which to sort.

See also

Array property, FieldName property, SortRequest event

FieldName property
(Windows only; GUI for .NET only)

Use this property in response to a SortRequest event to indicate the name of the field on which to sort the records in the ABL data source object.

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Data.SortRequestEventArgs class

If the field on which you are sorting is an array field, the ArrayIndex property is set to the 1-based index position of the array element on which to sort.

See also

Array property, FieldIndex property, SortRequest event

FileName property

Identifies the file name of the current `FileInputStream` or `FileOutputStream` object. The file name can include Unicode characters, and it can be an absolute pathname or a pathname relative to the current working directory.

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.IO.FileInputStream class, Progress.IO.FileOutputStream class

---

FileListId property

The `FileListId` property identifies the `FileList` VST. This VST temp-table has a single record which contains data about temp-table database. You can pass the `FileListId` property to the `GetVSTHandle( )` method to return the handle to the `FileList` VST. This property is read-only.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

Flush( ) method (FileOutputStream)

Flushes the file output stream buffer and forces any data to be written to the file.

Return type: VOID

Access: PUBLIC

Applies to: Progress.IO.FileOutputStream class

Syntax

```
Flush( )
```

Notes

- The AVM raises a runtime error if the file output stream is already closed.

Flush( ) method (OutputStream)

When overridden in a class derived from `Progress.IO.OutputStream`, this method's contract is that calling it indicates that any bytes previously buffered by the implementation of the output stream should be written to their destination.

Return type: VOID

Access: PUBLIC

Applies to: Progress.IO.OutputStream class

Syntax

```
Flush( )
```

Notes

- This method has no default behavior for the `File.IO.OutputStream` class.

Formatted property

Returns TRUE if the `JsonSerializer` instance is set to format the JSON string in a hierarchical manner using white space, carriage returns, and line feeds. Otherwise, it returns FALSE.

Data type: LOGICAL

Access: PUBLIC Read-only

Applies to: Progress.IO.JsonSerializer class

Get( ) method

Gets the value of the data member or property. This method can only be used to get the value of a public data member or property; otherwise, the method will raise an error.

Return type: The type of the data member

Access: PUBLIC

Applies to: Progress.Reflect.Variable class

Syntax

```
Get ( )

Get ( INPUT index AS INTEGER )

Get ( INPUT instance AS Progress.Lang.Object )

Get ( INPUT instance AS Progress.Lang.Object,
      INPUT index AS INTEGER )
```

index

For a data member or property that is an array, the index indicates which element to retrieve. If one of the overloads containing this parameter is invoked for a data member or property that is not an array, the method will raise an error.

instance

An instance of the class that contains the data member or property. This instance must be of the type specified by either the `DeclaringClass` or `OriginatingClass` property of the `Progress.Reflect.Variable` instance. Using the *instance* parameter with a static data member or property will raise an error.

Notes

- When applied to a property, the method will invoke the property's `GET` method, if there is one.
- Use the `Get( )` and `Get(`*index*`)` overloads to get the value of static data members and properties, and use the `Get(`*instance*`)` and `Get(`*instance*`, `*index*`)` overloads for instance data members and properties. Calling an overload on a data member or property of the incorrect scope will raise an error.
- When called on a data member or property that is an array, the `Get( )` and `Get(`*instance*`)` overloads return the entire array.
- The `Get(`*index*`)` and `Get(`*instance*`, `*index*`)` overloads cannot be used to retrieve individual elements from a .NET array.

GetCharacter( ) method (JsonArray)

Gets the JSON string value of one or more elements as a CHARACTER or CHARACTER array.

To get the string value of other JSON data types as a string, use the `GetJsonText( )` method.

Return type: CHARACTER

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetCharacter( INPUT index AS INTEGER )

GetCharacter( INPUT index AS INTEGER,
                    INPUT count AS INTEGER )
```

When this method is called with one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of *count* elements.

index

An INTEGER identifying the value of the element to be returned. Indexing into a JsonArrays is 1-based.

count

An INTEGER specifying the number of elements, starting with the element at *index*, to be returned to the ABL array.

A JSON error is raised if :

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1 or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a JSON string

GetCharacter( ) method (JsonObject)

Gets the JSON string value of the named property as a CHARACTER.

To get the value of other JSON data types as a string, use the `GetJsonText( )` method.

Return type: CHARACTER

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetCharacter( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression that denotes the name of the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a JSON string
- The property value does not fit into an ABL CHARACTER

GetClass( ) method (Progress.Lang.Class)

Returns the object reference for the Progress.Lang.Class instance associated with a specified class or interface type.

Return type: Progress.Lang.Class class

Access: PUBLIC STATIC

Applies to: Progress.Lang.Class class

Syntax

```
GetClass ( INPUT object-type-name AS CHARACTER )
```

object-type-name

A CHARACTER expression that evaluates to a fully qualified object type name.

If *object-type-name* does not evaluate to a valid ABL class or interface type, this method returns the Unknown value (?).

Note: Because the Progress.Lang.Class class has a private constructor, you cannot inherit from it. Therefore you must always invoke this static method using its class type name.

As an alternative to the GetClass() method, the GET-CLASS function also returns the object reference for the given class or interface, but it takes a type name as an argument instead of a CHARACTER expression.

See also

GET-CLASS function

GetClass( ) method (Progress.Lang.Object)

Returns the object reference for the Progress.Lang.Class instance associated with the current instance of a class.

Return type: Progress.Lang.Class class

Access: PUBLIC

Applies to: Progress.Lang.Object class

Syntax

```
GetClass( )
```

The Progress.Lang.Class instance returns type information on the class or interface type of the Progress.Lang.Object instance on which the method is executed.

GetClientPrincipal( ) method (Progress.Lang.OERequestInfo)

Returns a handle to a client principal object in an `OERequestInfo` instance, or the Unknown value (?).

Note: This method creates a deep copy of the client principal object that will remain in memory until you explicitly remove it in your ABL code.

Return type: HANDLE

Access: PUBLIC

Applies to: Progress.Lang.OERequestInfo class

Syntax

```
GetClientPrincipal( )
```

The bold code in the following fragment shows the assignment of a handle to the client principal in a server's `OERequestInfo` instance.

```
DEFINE VARIABLE hAppSrv      as HANDLE.
DEFINE VARIABLE hCPIn        as HANDLE.
DEFINE VARIABLE hCPOut       as HANDLE.
DEFINE VARIABLE ccid         as CHARACTER.
DEFINE VARIABLE bOK          as LOGICAL.
DEFINE VARIABLE loginState   as CHARACTER.

CREATE SERVER hAppSrv.
ccid = hAppSrv:REQUEST-INFO:clientContextID.

CREATE CLIENT-PRINCIPAL hCPIn.
hCPIn:INITIALIZE("userName@domainName", ccid, ?, "password").

hAppSrv:REQUEST-INFO:SetClientPrincipal(hCPIn).

bOK = hAppSrv:CONNECT("-URL http://hostname:port/apsv").

hCPOut = hAppSrv:RESPONSE-INFO:GetClientPrincipal().
IF VALID-HANDLE(hCPOut)
THEN loginState = hCPOut:LOGIN-STATE.
ELSE loginState = ?.

. . .

IF VALID-HANDLE(hCPOut)
   THEN DELETE OBJECT hCPOut.
```

See also

CURRENT-REQUEST-INFO attribute, CURRENT-RESPONSE-INFO attribute, LOCAL-VERSION-INFO attribute, REQUEST-INFO attribute, RESPONSE-INFO attribute, SetClientPrincipal( ) method

GetCOMHandle( ) method (JsonArray)

Gets the JSON number value of one or more elements as a COM-HANDLE or COM-HANDLE array.

Return type: COM-HANDLE

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetCOMHandle( INPUT index AS INTEGER )

GetCOMHandle( INPUT index AS INTEGER,
              INPUT count AS INTEGER )
```

When this method is called with one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of count elements.

index

An INTEGER identifying the value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An INTEGER specifying the number of elements, starting with the element at *index*, to be returned to ABL array.

A JsonError is raised if:

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1 or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a valid JSON integral number
- Any element does not fit into a COM-HANDLE

GetCOMHandle( ) method (JsonObject)

Gets the JSON number value of the named property as a COM-HANDLE.

Return type: COM-HANDLE

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetCOMHandle( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression that denotes the name of the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a valid JSON integral number
- The property value does not fit into a COM-HANDLE

GetConstructor( ) method

Returns a `Progress.Reflect.Constructor` instance describing the class constructor that matches the specified conditions. If there is no matching constructor, the method returns the Unknown value (?).

Return type: Progress.Reflect.Constructor class

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
GetConstructor ( INPUT params AS Progress.Lang.ParameterList )

GetConstructor ( INPUT flags AS Progrss.Reflect.Flags,
                 INPUT params AS Progress.Lang.ParameterList )
```

params

A `Progress.Lang.ParameterList` instance that describes the parameter list of the constructor you are looking for. If you are looking for a constructor that takes no parameters, pass a `ParameterList` instance with its `NumParameters` property set to 0.

flags

A `Progress.Reflect.Flags` instance indicating the access mode(s) the returned constructor must have. See Progress.Reflect.Flags enumeration for more information.

Caution: This method does not return static constructors, so setting the `Static` flag will raise an error. This method defaults to returning instance constructors, so you do not need to set the `Instance` flag.

Notes

- You only need to pass the access mode flags, as desired. This method never returns a constructor from a super class of the class it is applied to, and the method only returns instance constructors, so the `DeclaredOnly` and `Instance` flags are implied.
- `GetConstructor(`*`params`*`)` returns a `Progress.Reflect.Constructor` instance describing the default constructor if the `NumParameters` property of the `ParameterList` instance passed to it is set to 0 and no other constructor is defined. `GetConstructor(`*`params, flags`*`)` returns the default constructor under the same conditions if the constructor matches the specified flags.
- `GetConstructor(`*`params`*`)` only returns public constructors. `GetConstructor(`*`params, flags`*`)` returns all ABL class constructors based on the flags specified. However, you can only invoke constructors according to the same rules as when reflection is not used. For example, you can only invoke a protected constructor from within the class that defines it, or from a subclass.
- This method does not return .NET class constructors defined as private.

GetConstructors( ) method

Returns an array of `Progress.Reflect.Constructor` instances that match the specified conditions. There is no guaranteed order for the returned constructors. If there are no constructors that match the specified conditions, the method returns an indeterminate array.

Return type: Progress.Reflect.Constructor class EXTENT

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
GetConstructors ( )

GetConstructors ( INPUT flags AS Progress.Reflect.Flags )
```

`flags`

A `Progress.Reflect.Flags` instance indicating the access mode(s) the returned constructors must have. See Progress.Reflect.Flags enumeration for more information.

Caution: This method does not return static constructors, so setting the `Static` flag will raise an error. Unlike other reflection methods, `GetConstructors()` defaults to returning instance constructors, so you do not need to set the `Instance` flag for this method.

Notes

- You only need to pass the access mode flags, as desired. This method never returns a constructor from a super class of the class it is applied to, and the method only returns instance constructors, so the `DeclaredOnly` and `Instance` flags are implied.
- `GetConstructors( )` returns a `Progress.Reflect.Constructor` instance describing the default constructor if no other constructor is defined. `GetConstructors(`*`flags`*`)` returns the default constructor if no other constructor is defined and it matches the specified flags.

- `GetConstructors( )` only returns public constructors. `GetConstructors(`*`flags`*`)` returns all ABL class constructors based on the flags specified. However, you can only invoke constructors according to the same rules as when reflection is not used. For example, you can only invoke a protected constructor from within the class that defines it, or from a subclass.
- This method does not return .NET class constructors defined as private.

GetDate( ) method (JsonArray)

Gets the JSON string value of one or more elements as a DATE or DATE array.

Return type: DATE

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetDate( INPUT index AS INTEGER )

GetDate( INPUT index AS INTEGER,
         INPUT count AS INTEGER )
```

When this method is called with one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of count elements.

index

An INTEGER identifying the value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An INTEGER identifying the value of the element to be returned. Indexing into JsonArrays is 1-based.

A JsonError is raised if:

- *index* is less than 1, greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1, or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a JSON string
- Any element cannot be parsed as an ISO 8601 date in the `"YYYY-MM-DD"` format

GetDate( ) method (JsonObject)

Gets the JSON string value of the named property as a DATE.

Return type: DATE

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetDate( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression that denotes the name of the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a JSON string
- The property value cannot be parsed as an ISO 8601 date in the "YYYY-MM-DD" format

GetDatetime( ) method (JsonArray)

Gets the JSON string value of one or more elements as a DATETIME or DATETIME array.

Return type: DATETIME

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetDatetime( INPUT index AS INTEGER )

GetDatetime( INPUT index AS INTEGER,
             INPUT count AS INTEGER )
```

When this method is called with one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of count elements.

index

An integer value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An integer value specifying the number of elements, starting with the element at *index*, to be returned to the ABL array.

A JsonError is raised if:

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1 or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a JSON string
- Any element cannot be parsed as an ISO 8601 date and time in the format:

```
"YYYY-MM-DD[THH[:MM[:SS[.sss]]]]"
```

GetDatetime( ) method (JsonObject)

Gets the JSON string value of the named property as a DATETIME.

Return type: DATETIME

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetDatetime( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression that denotes the name of the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a JSON String
- The property value cannot be parsed as an ISO 8601 date and time in the format:

```
"YYYY-MM-DD[THH[:MM[:SS[.sss]]]]"
```

GetDatetimeTZ( ) method (JsonArray)

Gets the JSON string value of one or more elements as a DATETIME-TZ or DATETIME-TZ array.

Return type: DATETIME-TZ

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetDatetimeTZ( INPUT index AS INTEGER )

GetDatetimeTZ( INPUT index AS INTEGER,
                     INPUT count AS INTEGER )
```

When this method is called with one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of *count* elements.

index

An integer value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An integer value specifying the number of elements, starting with the element at *index*, to be returned to the ABL array.

A JsonError is raised if:

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1 or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a JSON String
- Any element cannot be parsed as an ISO 8601 date and time in this format:

  `"YYYY-MM-DD[THH[:MM[:SS[.sss]]]]Z"`

  Or this format:

  `"YYYY-MM-DD[THH[:MM[:SS[.sss]]]](+|-)HH:MM"`

GetDatetimeTZ( ) method (JsonObject)

Gets the JSON string value of the named property as a DATETIME-TZ.

Return type: DATETIME-TZ

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetDatetimeTZ( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression naming the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a JSON String.
- The property value cannot be parsed as an ISO 8601 date and time in this format:

```
"YYYY-MM-DD[THH[:MM[:SS[.sss]]]]Z"
```

Or this format:

```
"YYYY-MM-DD[THH[:MM[:SS[.sss]]]](+|-)HH:MM"
```

GetDecimal( ) method (JsonArray)

Gets the JSON number value of one or more elements as a DECIMAL or DECIMAL array.

Return type: DECIMAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetDecimal( INPUT index AS INTEGER )
GetDecimal( INPUT index AS INTEGER,
            INPUT count AS INTEGER )
```

When this method is called with one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of count elements.

index

An integer value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An integer value specifying the number of elements, starting with the element at *index*, to be returned in the ABL array.

A JsonError is raised if:

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)

- *count* is less than 1 or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a valid JSON number
- Any element is in scientific notation
- Any element does not fit into a DECIMAL value

GetDecimal( ) method (JsonObject)

Gets the JSON number value of the named property as a DECIMAL.

Return type: DECIMAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetDecimal( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression that denotes the name of the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a valid JSON number
- The property value is in scientific notation
- The property value does not fit into an ABL DECIMAL

GetEnum( ) method

Returns an enum instance of the specified enum member.

Return type: Enum type

Access: PUBLIC

Applies to: Progress.Reflect.AccessMode enumeration, Progress.Reflect.DataType enumeration, Progress.Reflect.Flags enumeration, Progress.Reflect.ParameterMode enumeration, any user-defined enum type

Syntax

```
GetEnum ( INPUT enum-member-name AS CHARACTER )

GetEnum ( INPUT value AS INT64 )
```

enum-member-name

The name of an enum member of the enum type the method is invoked on. For a flag enum type, this can be a comma-delimited list of enum members.

value

The underlying numeric value of an enum member of the enum type the method is invoked on. For a flag enum type, this value can represent two or more members.

The method returns an error if a name or numeric value does not correspond to a member of the enum type. The exception to this is that you can pass a value of 0 for a flag enum type even if no member has been explicitly defined for 0.

Both of the lines of code in the following example create an enum instance of type `Progress.Reflect.AccessMode` and assign `vAccess` the member `Public` :

```
DEFINE VARIABLE vAccess AS AccessMode.

vAccess = AccessMode:GetEnum("Public").
vAccess = AccessMode:GetEnum(1).
```

Notes

- This method is available for all built-in enums, and the compiler automatically generates it for all user-defined enums. For example, this code excerpt creates an instance of the user-defined `Permission` flag enum and assigns `vPerm` the members `Write` and `Create`:

```
ENUM Permission FLAGS:
    DEFINE ENUM None = 0
                Read = 0x01
                Write = 0x02
                ReadWrite = 0x03
                Create = 0x04
                Delete = 0x08.
END ENUM.
```

```
DEFINE VARIABLE vPerm AS Permission.

vPerm = Permission:GetEnum("Write,Create").
```

GetEnumName( ) method

Returns the name of the enumeration member associated with the specified numeric value. The method returns an error if invoked on a type that is not an enumeration or if passed the Unknown value (?).

Return type: CHARACTER

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
GetEnumName ( INPUT value AS INT64 )
```

value

The underlying numeric value of an enum member. If no member is defined for the value, the method returns the Unknown value (?).

Notes

- For a flag enum instance that represents multiple flags set, the method returns a comma-separated list of members, unless there is a member that is explicitly defined as a combination of those members. For example, for the `Permission` enum type shown below, specifying the value 3 would return "ReadWrite" instead of "Read,Write".

```
ENUM Permission FLAGS:
    DEFINE ENUM None = 0
                Read = 0x01
                Write = 0x02
                Update = 0x02
                ReadWrite = 0x03
                Create = 0x04
                Delete = 0x08.
END ENUM.
```

- If the enum type defines multiple members for the same value, as the `Permission` enum type above does for `Write` and `Update`, the method returns one of the member names, but there is no guarantee as to which one will be returned.

GetEnumNames( ) method

Returns a comma-separated list of names of all the members of the enum type. The method returns an error if invoked on a type that is not an enumeration.

Return type: CHARACTER

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
GetEnumNames ( )
```

Notes

- The list of names returned is in order of ascending numerical values, regardless of whether the values were explicitly set. For flag enum types, the members with positive numeric values are listed in ascending order first, and then the members with negative numeric values are listed next, also in ascending order.
- If two or more members are defined with the same numeric value, there is no guarantee for the order in which they are returned.

GetEnumValue( ) method

Returns the underlying numeric value of the specified enum member. The method returns an error if invoked on a type that is not an enumeration or if passed the Unknown value (?).

Return type: INT64

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
GetEnumValue ( INPUT enum-member-name AS CHARACTER )
```

enum-member-name

The name of an enum member of the enum type the method is invoked on. For a flag enum type, this can be a comma-delimited list of names. If no member is defined for the specified name, or any of the names in a list is not defined, the method returns the Unknown value (?).

GetEnumValues( ) method

Returns a comma-separated list of all the underlying numeric values (as a string) of an enumeration type. The method returns an error if invoked on a type that is not an enumeration.

Return type: CHARACTER

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
GetEnumValues ( )
```

Notes

- The list of values is returned in ascending order. For flag enum types, the positive numeric values are listed in ascending order first, and then the negative numeric values are listed next, also in ascending order.

GetEvent( ) method

Returns a `Progress.Reflect.Event` instance describing the class event that matches the specified conditions. If there is no matching event, the method returns the Unknown value (?).

Return type: Progress.Reflect.Event class

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
GetEvent ( INPUT event-name AS CHARACTER )

GetEvent ( INPUT event-name AS CHARACTER,
           INPUT flags AS Progress.Reflect.Flags )
```

event-name

A CHARACTER expression that evaluates to the name of the event.

flags

A `Progress.Reflect.Flags` instance indicating the access mode(s), scope(s), and/or class level(s) the returned event must have. See Progress.Reflect.Flags enumeration for more information.

Notes

- If there are one or more overrides of any particular event, the `Progress.Reflect.Event` instance describes the override from the most derived subclass that overrides that event.
- `GetEvent(`*`event-name`*`)` only returns public, non-static events. `GetEvent(`*`event-name, flags`*`)` returns all ABL class events based on the flags specified. However, you can only invoke events according to the same rules as when reflection is not used. For example, you can only invoke a protected event from within the class that defines it, or from a subclass.
- This method does not return .NET class events defined as private or internal.

GetEvents( ) method

Returns an array of `Progress.Reflect.Event` instances describing the class events that match the specified conditions. There is no guaranteed order for the returned events. If there are no events that match the specified conditions, the method returns an indeterminate array.

Return type: Progress.Reflect.Event class EXTENT

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
GetEvents ( )

GetEvents ( INPUT flags AS Progress.Reflect.Flags )
```

flags

A `Progress.Reflect.Flags` instance indicating the access mode(s), scope(s), and/or class level(s) the returned events must have. See Progress.Reflect.Flags enumeration for more information.

Notes

- If there are one or more overrides of any particular event, the `Progress.Reflect.event` instance describes the override from the most derived subclass that overrides that event. You will NOT get multiple entries for overridden events. See GetMethod( ) method for an example of how overridden class members are handled.
- `GetEvents( )` only returns public, non-static events. `GetEvents(flags)` returns all ABL class events based on the flags specified. However, you can only invoke events according to the same rules as when reflection is not used. For example, you can only invoke a protected event from within the class that defines it, or from a subclass.
- This method does not return .NET class events defined as private or internal.

GetHandle( ) method (JsonArray)

Gets the JSON number value of one or more elements as a HANDLE or HANDLE array.

Return type: HANDLE

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetHandle( INPUT index AS INTEGER )

GetHandle( INPUT index AS INTEGER,
                 INPUT count AS INTEGER )
```

When this method is called with one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of count elements.

index

An integer value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An integer value specifying the number of elements, starting with the element at *index*, to be returned to the ABL array.

A JsonError is raised if:

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1 or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a valid JSON integral number
- Any element does not fit into an ABL HANDLE

GetHandle( ) method (JsonObject)

Gets the JSON number value of the named property as a HANDLE.

Return type: HANDLE

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetHandle( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression that indicates the name of the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a valid JSON integral number
- The property value does not fit into an ABL HANDLE

GetHandlerParameters( ) method

Returns an array of `Progress.Reflect.Parameter` instances describing the parameters for the given event handler. If there are no parameters, the method returns the indeterminate array.

Return type: Progress.Reflect.Parameter EXTENT

Access: PUBLIC

Applies to: Progress.Reflect.Event class

Syntax

```
GetHandlerParameters ( )
```

GetIndexInfoByID( ) method

Returns information about a temp-table index when the ID is specified. The ID is a numeric value that identifies the index. If the ID is valid for an existing temp-table index, this returns TRUE. Otherwise, it returns FALSE. This property is read-only.

Return type: LOGICAL

Access: PUBLIC STATIC

Applies to: Progress.Database.TempTableInfo class

Syntax

This method supports two overloaded versions. The first version returns the index name and name of the procedure or class that instantiated the temp-table.

```
GetIndexInfoByID( INPUT index-id AS INTEGER,
                  OUTPUT index-name AS CHARACTER,
                  OUTPUT proc-name AS CHARACTER )
```

The second version returns the index-name, the handle to the temp-table for that index, and the name of the procedure or class that instantiated the temp-table.

```
GetIndexInfoByID( INPUT index-id AS INTEGER,
                  OUTPUT index-name AS CHARACTER,
                  OUTPUT table-handle AS HANDLE,
                  OUTPUT proc-name AS CHARACTER )
```

index-id

An integer expression which evaluates the ID of the temp-table index.

index-name

Returns the name of the index specified by the index ID. The format is `tablename.indexname`. For example: `ttCustomer.custnum`.

table-handle

Returns a HANDLE to the temp-table specified by the index ID.

proc-name

Returns the name of the procedure or class where the temp-table was created. For a static temp-table, the procedure is used with the DEFINE TEMP-TABLE statement. For a dynamic temp-table, it is the procedure that executes the TEMP-TABLE-PREPARE( ) method. The name of the procedure is the name that is specified when you run mydir/test.p.

GetIndexStatHistoryHandle( ) method

Returns the handle of the temp-table used for storing the archived statistics from the _IndexStat virtual system table. The property returns the Unknown value (?) if the ArchiveIndexStatistics property is never set to TRUE.

Return type: HANDLE

Access: PUBLIC STATIC

Applies to: Progress.Database.TempTableInfo class

Syntax

```
GetIndexStatHistoryHandle()
```

The temp-table that the method returns is a dynamic temp-table called _IndexStatHistory. This is maintained by the AVM and cannot be deleted when the ArchiveIndexStatistics property is set to TRUE. If you try to delete it, the AVM raises an error. The _IndexStatHistory dynamic temp-table can be deleted if the ArchiveIndexStatistics property is set to FALSE.

The records in this table can be manipulated by emptying the temp-table. If the ArchiveIndexStatistics property is again set to TRUE, then the dynamic temp-table _IndexStatHistory is recreated. It contains all the fields from _IndexStat, except the _IndexStat-id field.

The fields that will be present in the _IndexStat table are listed below:

_Index-name

A character field with the index name in the format table-name.index-name.

_Proc-name

A character field with the name of the procedure where the temp-table was created.

_*Delete*-Timestamp

The date-time field with the timestamp denoting when the temp-table index got deleted.

GetInteger( ) method (JsonArray)

Gets the JSON number value of one or more elements as an INTEGER or INTEGER array.

Return type: INTEGER

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetInteger( INPUT index AS INTEGER )

GetInteger( INPUT index AS INTEGER,
            INPUT count AS INTEGER )
```

When this method is called with one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of count elements.

index

An integer value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An integer value specifying the number of elements, starting with the element at *index*, to be returned in the ABL array.

A JsonError is raised if:

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1 or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a valid JSON integral number
- Any element does not fit into an ABL INTEGER

GetInteger( ) method (JsonObject)

Gets the JSON number value of the named property as an INTEGER

Return type: INTEGER

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetInteger( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression that indicates the name of the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a valid JSON integral number
- The property value does not fit into an ABL INTEGER

GetInterfaces( ) method

Returns an array of `Progress.Lang.Class` instances describing the interfaces implemented by the class. There is no guaranteed order for the interfaces returned. If there are no interfaces, the method returns an indeterminate array.

Return type: Progress.Lang.Class EXTENT

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
GetInterfaces ( )
```

Notes

- The method returns any interface implemented by the class, any interface implemented by any of its super classes, and any super interfaces those interfaces inherit from.

GetInt64( ) method (JsonArray)

Gets the JSON number value of one or more elements as an INT64 or INT64 array.

Return type: INT64

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetInt64( INPUT index AS INTEGER )

GetInt64( INPUT index AS INTEGER,
          INPUT count AS INTEGER )
```

If the method returns one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of count elements.

index

An integer value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An INTEGER specifying the number of elements, starting with the element at index, to be returned in the ABL array.

A JsonError is raised if:

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1 or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a valid JSON integral number
- Any element does not fit into an ABL INT64

GetInt64( ) method (JsonObject)

Gets the JSON number value of the named property as an INT64.

Return type: INT64

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetInt64( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression that denotes the name of the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a valid JSON integral number
- The property value does not fit into an ABL INT64

GetJsonArray( ) method (JsonArray)

Gets the JsonArray value of one or more elements as a JsonArray or array of JsonArray.

Return type: Progress.Json.ObjectModel.JsonArray class

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetJsonArray( INPUT index AS INTEGER )

GetJsonArray( INPUT index AS INTEGER,
                    INPUT count AS INTEGER )
```

When this method is called with one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of count elements.

index

An INTEGER identifying the value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An INTEGER specifying the number of elements, starting with the element at *index*, to be returned in the ABL array.

A JsonError is raised if:

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1 or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a JsonArray

The following example demonstrates the functionality of the `GetJSonArray( )` method.

The following is a sample JsonArray:

```
[
    {"OrderNum": 82, "OrderDate": "2010-09-20"},
    {"OrderNum": 83, "OrderDate": "2010-09-20"},
    ["Q1", "Q3", 15, 1980, null, 2010]
]
```

The following code fragment demonstrates the use of `GetJsonArray( )` method:

```
DEFINE VARIABLE OrderInfo AS LONGCHAR NO-UNDO.
DEFINE VARIABLE OrderArray AS JsonArray NO-UNDO.
DEFINE VARIABLE SummaryArray AS JsonArray NO-UNDO.
DEFINE VARIABLE myLongchar as LONGCHAR NO-UNDO.

/*
 .
 . Set OrderInfo to the JsonArray shown above.
 .
 */

OrderArray = CAST(myParser:Parse(OrderInfo), JsonArray).
SummaryArray = OrderArray:GetJsonArray(3).
SummaryArray:Write(myLongchar, TRUE).
```

The following is the resulting `myLongchar` value:

```
["Q1", "Q3", 15, 1980, null, 2010]
```

GetJsonArray( ) method (JsonObject)

Gets the JsonArray value of the named property.

Return type: Progress.Json.ObjectModel.JsonArray class

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetJsonArray( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression indicates the name of the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a JsonArray

The following example demonstrates the functionality of the `GetJSonArray( )` method.

The following is a sample JsonObject:

```
{
  "CustOrder": {
    "CustNum": 1,
    "Name": "Lift Tours",
    "NewCust": false,
    "ttOrder": [
      {"OrderNum": 82, "OrderDate": "2010-09-20"},
      {"OrderNum": 83, "OrderDate": "2010-09-20"}
        ]
    }
}
```

The following is a code fragment that uses the `GetJsonArray( )` method:

```
DEFINE VARIABLE CustInfo AS LONGCHAR NO-UNDO.
DEFINE VARIABLE myObj AS JsonObject NO-UNDO.
DEFINE VARIABLE myArray AS JsonArray NO-UNDO.
DEFINE VARIABLE myLongchar as LONGCHAR NO-UNDO.

/*
 .
 . Set CustInfo to JsonObject shown above.
 .
 */

myObj = CAST(myParser:Parse(CustInfo), JsonObject).
myArray = myObj:GetJsonObject("CustOrder"):GetJsonArray("ttOrder").
myArray:Write(myLongchar, TRUE).
```

The following is the resulting `myLongchar` value:

```
[
    {"OrderNum": 82, "OrderDate": "2010-09-20"},
    {"OrderNum": 83, "OrderDate": "2010-09-20"}
]
```

GetJsonObject( ) method (JsonArray)

Gets the JsonObject value of one or more elements as a JsonObject, or as an array of JsonObjects.

Return type: Progress.Json.ObjectModel.JsonObject class

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetJsonObject( INPUT index AS INTEGER )

GetJsonObject( INPUT index AS INTEGER,
               INPUT count AS INTEGER )
```

When this method is called with one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of count elements.

index

An INTEGER identifying the value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An INTEGER specifying the number of elements, starting with the element at *index*, to be returned in the ABL array.

A JsonError is raised if:

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a JsonObject

The following example demonstrates the functionality of the `GetJSonObject( )` method.

The following is a sample JsonArray:

```
[
    {"OrderNum": 82, "OrderDate": "2010-09-20"},
    {"OrderNum": 83, "OrderDate": "2010-09-20"}
]
```

The following is a code fragment that uses the `GetJsonObject( )` method:

```
DEFINE VARIABLE OrderInfo AS LONGCHAR NO-UNDO.
DEFINE VARIABLE OrderArray AS JsonArray NO-UNDO.
DEFINE VARIABLE myOrder AS JsonObject NO-UNDO.
DEFINE VARIABLE myLongchar as LONGCHAR NO-UNDO.

/*
 .
 . Set OrderInfo to the JsonArray shown above.
 .
 */

OrderArray = CAST(myParser:Parse(OrderInfo), JsonArray).
myOrder = OrderArray:GetJsonObject(2).
myOrder:Write(myLongchar, TRUE).
```

The following is the resulting `myLongchar` value:

```
{"OrderNum": 83, "OrderDate": "2010-09-20"}
```

GetJsonObject( ) method (JsonObject)

Gets the JsonObject value of the named property.

Return type: Progress.Json.ObjectModel.JsonObject class

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetJsonObject( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression that indicates the name the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a JsonObject

The following example demonstrates the functionality of the `GetJSonObject( )` method.

The following is a sample JsonObject:

```
{
  "CustOrder": {
    "CustNum": 1,
    "Name": "Lift Tours",
    "NewCust": false,
    "Order": {"OrderNum": 82, "OrderDate": "2010-09-20"}
    ]
  }
}
```

The following code fragment uses the `GetJsonObject( )` method:

```
DEFINE VARIABLE CustInfo AS LONGCHAR NO-UNDO.
DEFINE VARIABLE myObj AS JsonObject NO-UNDO.
DEFINE VARIABLE myOrderObj AS JsonObject NO-UNDO.
DEFINE VARIABLE myLongchar as LONGCHAR NO-UNDO.

/*
 .
 . Set CustInfo to the JsonObjet shown above.
 .
 */

myObj = CAST(myParser:Parse(CustInfo), JsonObject).
myOrderObj = myObj:GetJsonObject("CustOrder"):GetJsonObject("Order").
myOrderObject:Write(myLongchar, TRUE).
```

The following is the resulting `myLongchar` value:

```
{"OrderNum": 82, "OrderDate": "2010-09-20"}
```

GetJsonText( ) method (JsonArray)

Gets one or more elements as a LONGCHAR or LONGCHAR array, regardless of the JSON data type.

The following are the type of return values:

- If the element is a simple type (string, number, boolean or null), it returns the value as it appears in the JsonArray as a string. For example, "abc", "123.45", "-13E+20", "true", "false", and "null".
- A JsonObject or a JsonArray element is returned in its serialized form. That is, as a LONGCHAR representing all the values it contains. For example, {"name":"Alfred","age":32} or ["jump rope", 17].

The method is useful especially in situations where JSON number values cannot be represented by any of the ABL numeric data types (INTEGER, INT64 and DECIMAL). This method allows you to retrieve values regardless of the JSON data type. `GetCharacter( )` and `GetLongchar( )` require that the JSON value be a JSON string.

Return type: LONGCHAR

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetJsonText( )

GetJsonText( INPUT index AS INTEGER )

GetJsonText( INPUT index AS INTEGER,
             INPUT count AS INTEGER )
```

When this method is called with one parameter, the value of a single element is returned. Using no parameters directs the AVM to return the value of the entire JsonArray. Using two parameters directs the AVM to return an ABL array consisting of count elements.

index

An INTEGER identifying the value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An INTEGER specifying the number of elements, starting with the element at *index*, to be returned in the ABL array.

A JsonError is raised if:

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1 or is the Unknown value (?)

- The sum of *index* and *count* is greater than the length of the JsonArray
- The JSON value contains a character that cannot be code page converted to `-cpinternal`

GetJsonText( ) method (JsonObject)

Gets the named property as a LONGCHAR, regardless of the JSON data type. The following are the type of return values:

- If the property value is a simple type (string, number, boolean or null), it returns the value as it appears in the JsonObject as a string. For example, "abc", "123.45", "-13E+20", "true", "false", and "null".
- A JsonObject or JsonArray property is returned in its serialized form. That is, as a LONGCHAR representing all contained values. For example, {"name":"Alfred","age":32} or ["jump rope", 17].

The method is useful especially for situations where JSON number values cannot be represented by any of the ABL numeric data types (INTEGER, INT64 and DECIMAL). This method allows you to retrieve values regardless of the JSON data type. `GetCharacter( )` and `GetLongchar( )` require that the JSON value be a JSON string.

Return type: LONGCHAR

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetJsonText( )

GetJsonText( INPUT property-name AS CHARACTER )
```

With one parameter, the value of the named property is returned. With no parameters the value of the entire JsonObject is returned.

property-name

A optional CHARACTER expression that denotes the name of the property value to be retrieved from the JsonObject. The default is to return the entire object.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value contains a character that cannot be code page converted to `-cpinternal`

GetLogical( ) method (JsonArray)

Gets the JSON boolean value of one or more elements as a LOGICAL or LOGICAL array.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetLogical( INPUT index AS INTEGER )

GetLogical( INPUT index AS INTEGER,
            INPUT count AS INTEGER )
```

When this method is called with one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of *count* elements.

index

An INTEGER identifying the value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An INTEGER specifying the number of elements, starting with the element at *index*, to return in the ABL array.

A JsonError is raised if:

- *count* is less than 1 or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a JSON boolean.

GetLogical( ) method (JsonObject)

Gets the JSON boolean value of the named property as a LOGICAL.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetLogical( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression that denotes the name of the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a JSON boolean

GetLongchar( ) method (JsonArray)

Gets the JSON string value of one or more elements as a LONGCHAR or LONGCHAR array.

To get the value of other JSON data types as a string, use the `GetJsonText( )` method.

Return type: LONGCHAR

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetLongchar ( INPUT index AS INTEGER )

GetLongchar ( INPUT index AS INTEGER,
              INPUT code-page AS CHARACTER )

GetLongchar ( INPUT index AS INTEGER,
              INPUT count AS INTEGER )

GetLongchar ( INPUT index AS INTEGER )
              INPUT count AS INTEGER )
              INPUT code-page AS CHARACTER )
```

If *code-page* is specified, the returned LONGCHAR will be set to that *code-page*. By default, the returned LONGCHAR will be set to `-cpinternal`.

When this method is called without the *count* parameter, the value of a single element is returned. Using the *count* parameter directs the AVM to return an ABL array consisting of count elements.

index

An INTEGER identifying the value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An INTEGER specifying the number of elements, starting with the element at index, to be returned in the ABL array.

If *code-page* is specified, the returned LONGCHAR is set to that code-page. By default, the returned LONGCHAR is set to `-cpinternal`. To get the string value of other JSON data types use the `GetJsonText( )` method.

A JsonError is raised if:

- `index` is less than 1, greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1 or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray

- Any element is not a JSON String

GetLongchar( ) method (JsonObject)

Gets the JSON string value of the named property as a LONGCHAR.

To get the value of other JSON data types as a string, use the `GetJsonText( )` method.

Return type: LONGCHAR

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetLongchar( INPUT property-name AS CHARACTER )

GetLongchar( INPUT property-name AS CHARACTER,
             INPUT code-page AS CHARACTER )
```

If *code-page* is specified, the returned LONGCHAR is set to that code-page. By default, the returned LONGCHAR is set to `-cpinternal`.

property-name

A CHARACTER expression that denotes the name of the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

code-page

A CHARACTER expression specifying the code page to which the returned LONGCHAR should be set. If this parameter is the empty string ("") or is the Unknown value (?) it defaults to `-cpinternal`.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- *code-page* is not supported by OpenEdge
- The property does not exist
- The property value is not a JSON string

GetMemptr( ) method (JsonArray)

Gets the JSON string value of one or more elements as a MEMPTR or MEMPTR array.

Return type: MEMPTR

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetMemptr( INPUT index AS INTEGER )

GetMemptr( INPUT index AS INTEGER,
           INPUT count AS INTEGER )
```

With one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of count elements.

index

An INTEGER identifying the value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An INTEGER specifying the number of elements, starting with the element at *index*, to be returned in the ABL array.

A JsonError is raised if:

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1 or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a JSON string
- Any element cannot be parsed as a Base-64 encoded string

GetMemptr( ) method (JsonObject)

Gets the JSON string value of the named property as a MEMPTR.

Return type: MEMPTR

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetMemptr( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression that indicates the name of the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)

- The property does not exist
- The property value is not a JSON string
- The property value cannot be parsed as a Base-64 encoded string

GetMessage( ) method (Class)

Returns the error message for the indexed error in the error object. If there is no error message at the indicated index, the method returns the empty string.

Return type: CHARACTER

Access: PUBLIC

Applies to: Progress.Lang.ProError class, System.Exception class (OpenEdge-enhanced)

Syntax

```
GetMessage (INPUT MessageIndex AS INTEGER)
```

MessageIndex

The NumMessages property indicates the number of error message and number pairs in an error object. Specify an index between 1 and the value of the NumMessages property.

For a .NET System.Exception class, the information from the current object reference is used to populate message 1. If there is an InnerException, its information is used to populate message 2. This process continues down through any other nested InnerExceptions. The format of each message is:

```
Exception-Typename:Message
```

For example, if you pass the value 2 to access the third member of a control collection and there are less than three elements in the collection you will get this message:

```
System.ArgumentOutOfRangeException: Index 2 is out of range
```

The string "Index 2 is out of range" is the value of the Exception Message property.

See also

NumMessages property, GetMessageNum( ) method

GetMessageNum( ) method

Returns the error message number associated with the indexed error in the error object.

Return type: INTEGER

Access: PUBLIC

Applies to: Progress.Lang.ProError class

Syntax

```
GetMessageNum (INPUT MessageIndex AS INTEGER)
```

MessageIndex

The NumMessages property indicates the number of error message and number pairs in an error object. Specify an index between 1 and the value of the NumMessages property.

For Progress.Lang.SysError objects, the method returns the OpenEdge message number for the system generated error. For Progress.Lang.AppError objects, the application message number returned is the number provided to the AppError( ) constructor or the AddMessage( ) method. If there is no error message at the index, the method returns the empty string.

.NET Exception objects also do not use this method.

See also

NumMessages property, GetMessage( ) method (Class)

GetMethod( ) method

Returns a Progress.Reflect.Method instance describing the class method that matches the specified conditions. If there is no matching method, the method returns the Unknown value (?).

Return type: Progress.Reflect.Method class

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
GetMethod ( INPUT method-name AS CHARACTER,
            INPUT params AS Progress.Lang.ParameterList )

GetMethod ( INPUT method-name AS CHARACTER,
            INPUT flags AS Progress.Reflect.Flags,
            INPUT params AS Progress.Lang.ParameterList )
```

method-name

A CHARACTER expression that evaluates to the name of the method.

params

A Progress.Lang.ParameterList instance that describes the parameter list of the method you are looking for. If you are looking for a method that takes no parameters, pass a ParameterList instance with its NumParameters property set to 0.

flags

A Progress.Reflect.Flags instance indicating the access mode(s), scope(s), and/or class level(s) the returned method must have. See Progress.Reflect.Flags enumeration for more information.

Notes

- If there are one or more overrides of the specified method, GetMethod( ) returns the override from the most derived subclass that overrides that method. (In other words, GetMethod( ) returns the version of the method that would be called if you invoked the method on that class.) For example:

```
CLASS ClassA:
  METHOD PUBLIC VOID MethodA:
    ...
  END METHOD.
END CLASS.

CLASS ClassB INHERITS ClassA:
  METHOD PUBLIC OVERRIDE VOID MethodA:
    ...
  END METHOD.
END CLASS.

CLASS ClassC INHERITS ClassB:
END CLASS.
```

  Calling GetMethod("MethodA", paramlistForZeroParams) on an instance of Progress.Lang.Class derived from ClassC returns a Progress.Reflect.Method instance describing the MethodA override defined in ClassB. In other words, the DeclaringClass property would be set to ClassB.

- GetMethod(*method-name, params*) only returns public, non-static methods. GetMethod(*method-name, params, flags*) returns all ABL class methods based on the flags specified. However, you can only invoke methods according to the same rules as when reflection is not used. For example, you can only invoke a protected method from within the class that defines it, or from a subclass.
- This method does not return .NET class methods defined as private or internal.

GetMethods( ) method

Returns an array of Progress.Reflect.Method instances describing the class methods that match the specified conditions. There is no guaranteed order for the returned methods. If there are no methods that match the specified conditions, the method returns an indeterminate array.

Return type: Progress.Reflect.Method class EXTENT

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
GetMethods ( )

GetMethods ( INPUT flags AS Progress.Reflect.Flags )
```

flags

A `Progress.Reflect.Flags` instance indicating the access mode(s), scope(s), and/or class level(s) the returned method must have. See Progress.Reflect.Flags enumeration for more information.

Notes

- If there are one or more overrides of any particular method, the `Progress.Reflect.Method` instance describes the override from the most derived subclass that overrides that method. You will NOT get multiple entries for overridden methods. See GetMethod( ) method for more information on how overridden methods are handled.
- `GetMethods( )` only returns public, non-static methods. `GetMethods(`*flags*`)` returns all ABL class methods based on the flags specified. However, you can only invoke methods according to the same rules as when reflection is not used. For example, you can only invoke a protected method from within the class that defines it, or from a subclass.
- This method does not return .NET class methods defined as private or internal.

GetNames( ) method (JsonObject)

Returns an ABL array that can be used to access all properties in JsonObject.

Return type: CHARACTER

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetNames( )
```

If the object is empty, an array is returned whose extent is the Unknown value (?). To examine every property in an object, loop through the returned ABL array and retrieve each property using its name. Note that property names are case-sensitive.

GetParameters( ) method

Returns an array of `Progress.Reflect.Parameter` instances describing the parameters for the given constructor or method. If there are no parameters, the method returns an indeterminate array.

Return type: Progress.Reflect.Parameter EXTENT

Access: PUBLIC

Applies to: Progress.Reflect.Constructor class, Progress.Reflect.Method class

Syntax

```
GetParameters ( )
```

GetProperties( ) method

Returns an array of `Progress.Reflect.Property` instances describing the class properties that match the specified conditions. There is no guaranteed order for the returned properties. If there are no properties that match the specified conditions, the method returns an indeterminate array.

Return type: Progress.Reflect.Property class EXTENT

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
GetProperties ( )

GetProperties ( INPUT flags AS Progress.Reflect.Flags )
```

flags

A `Progress.Reflect.Flags` instance indicating the access mode(s), scope(s), and/or class level(s) that the returned properties must have. See Progress.Reflect.Flags enumeration for more information.

Notes

- If there are one or more overrides of any particular property, the `Progress.Reflect.Property` instance describes the override from the most derived subclass that overrides that property. You will NOT get multiple entries for overridden properties. See GetMethod( ) method for an example of how overridden class members are handled.
- `GetProperties( )` only returns public, non-static properties. `GetProperties(`*`flags`*`)` returns all ABL class properties based on the flags specified. However, you can only get or set properties according to the same rules as when reflection is not used. For example, you can only get or set a protected property from within the class that defines it, or from a subclass.
- This method does not return .NET class properties defined as private or internal.

GetProperty( ) method

Returns a `Progress.Reflect.Property` instance describing the class property that matches the specified conditions. If there is no matching property, the method returns the Unknown value (?).

Return type: Progress.Reflect.Property class

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
GetProperty ( INPUT property-name AS CHARACTER )

GetProperty ( INPUT property-name AS CHARACTER,
              INPUT flags AS Progress.Reflect.Flags )
```

property-name

A CHARACTER expression that evaluates to the name of the property.

flags

A `Progress.Reflect.Flags` instance indicating the access mode(s), scope(s), and/or class level(s) the returned property must have. See Progress.Reflect.Flags enumeration for more information.

Notes

- If there are one or more overrides of any particular property, the `Progress.Reflect.Property` instance describes the override from the most derived subclass that overrides that property. See GetMethod( ) method for an example of how overridden class members are handled.
- `GetProperty(`*`property-name`*`)` only returns public, non-static properties. `GetProperty(`*`property-name, flags`*`)` returns all ABL class properties based on the flags specified. However, you can only get or set properties according to the same rules as when reflection is not used. For example, you can only get or set a protected property from within the class that defines it, or from a subclass.
- This method does not return .NET class properties defined as private or internal.

GetPropertyValue( ) method

Gets a property's value at run time. Use the `GetPropertyValue( )`method to access a property's value dynamically, even if you do not know the property's name or type when you are using the method at compile time.

Return type: Any data type

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

Four overloaded versions are supported. The first version operates on a scalar instance property, where the property is any data type that is not, itself, an array.

```
[ return-value = ] class-object-reference:GetPropertyValue (
  INPUT object-reference AS Progress.Lang.Object ,
  INPUT property-name AS CHARACTER )
```

The second version operates on a scalar static property.

```
[ return-value = ] class-object-reference:GetPropertyValue (
  INPUT property-name AS CHARACTER )
```

The third version operates on an instance array or .NET indexed property.

```
[ return-value = ] class-object-reference:GetPropertyValue (
  INPUT object-reference AS Progress.Lang.Object,
  INPUT property-name AS CHARACTER, INPUT index AS any-data-type )
```

The fourth version operates on a static array or .NET indexed property.

```
[ return-value = ] class-object-reference:GetPropertyValue (
  INPUT object-reference AS Progress.Lang.Object,
  INPUT property-name AS CHARACTER, INPUT index AS any-data-type )
```

return-value

Specifies an ABL data element, that is assigned the value returned from the property's GET accessor. This return value can have any valid ABL or .NET method return type. The AVM automatically does the required conversions as if there were a CAST or an ABL conversion function such as STRING or INTEGER present.

class-object-reference

Specifies a reference to an instance of `Progress.Lang.Class` class; this must be a type that exposes the property. For instance properties, *object-reference* and *class-object-reference* must be the same type or one must inherit from the other, and both must either define the property or inherit it.

object-reference

Specifies a reference to an ABL or .NET class instance that exposes the specified property as an instance member. The compiler allows `object-reference` to be declared as any object type. At run time, the object type must resolve to the type that exposes the property.

Note: The overloads of `GetPropertyValue( )`that take an object reference can be used with a static property by setting object-reference to the Unknown value (?). Alternatively, use the overloads that are designed specifically for static properties.

Note: A run-time error arises if *object-reference* does not inherit from *class-object-reference*, or vice-versa.

---

property-name

Specifies a CHARACTER name of the property. The AVM evaluates *property-name* at run time. Note that the property cannot be defined in a built-in type.

index

Specifies an integer expression for the index of the desired element. If the property is a .NET indexed property, index is an expression for the specified element. The index for an indexed property does not have to be an integer.

any-data-type

Specifies the data type of the associated element. The set of possible data types depends on the element. For example, the index of an ABL array element can only be an integer type. However, the return type of a property or array property element can be any ABL primitive data or object type.

See also

DYNAMIC-PROPERTY function, SetPropertyValue( ) method

GetRaw( ) method (JsonArray)

Gets the JSON string value of one or more elements as a RAW or RAW array.

Return type: RAW

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetRaw( INPUT index AS INTEGER )

GetRaw( INPUT index AS INTEGER,
        INPUT count AS INTEGER )
```

When this method is called with one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of `count` elements.

index

An INTEGER identifying the value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An INTEGER specifying the number of elements, starting with the element at *index*, to be returned in the ABL array.

A JsonError is raised if:

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1 or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a JSON string
- Any element cannot be parsed as a Base-64 encoded string

GetRaw( ) method (JsonObject)

Gets the JSON string value of the named property as a RAW.

Return type: RAW

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetRaw( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression that denotes the name of the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a JSON string
- The property value cannot be parsed a Base-64 encoded string

GetRecid( ) method (JsonArray)

Gets the JSON number value of one or more elements as a RECID or RECID array.

Return type: RECID

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetRecid( INPUT index AS INTEGER )

GetRecid( INPUT index AS INTEGER,
          INPUT count AS INTEGER )
```

When this method is called with one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of count elements.

index

An INTEGER identifying the value of the element to be returned. Indexing into JsonArrays is 1-based.

An INTEGER specifying the number of elements, starting with the element at *index*, to be returned in the ABL array.

A JsonError is raised if:

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1 or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a valid JSON integral number
- Any element does not fit an ABL RECID

GetRecid( ) method (JsonObject)

Gets the JSON number value of the named property as a RECID.

Return type: RECID

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetRecid( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression naming the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a valid JSON integral number
- The property value does not fit into an ABL RECID

GetRowid( ) method (JsonArray)

Gets the JSON string value of one or more elements as a ROWID or ROWID array.

Return type: ROWID

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetRowid( INPUT index AS INTEGER )

GetRowid( INPUT index AS INTEGER,
          INPUT count AS INTEGER )
```

This method when called with one parameter, the value of a single element is returned. Using two parameters directs the AVM to return an ABL array consisting of `count` elements.

index

An INTEGER identifying the value of the element to be returned. Indexing into JsonArrays is 1-based.

count

An INTEGER specifying the number of elements, starting with the element at `index`, to be returned in the ABL array.

A JsonError is raised if:

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1 or is the Unknown value (?)
- The sum of *index* and *count* is greater than the length of the JsonArray
- Any element is not a JSON string
- Any element cannot be parsed as a Base-64 encoded string.

GetRowid( ) method (JsonObject)

Gets the JSON string value of the named property as a ROWID.

Return type: ROWID

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetRowid( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression that denotes the name of the property value to be retrieved from the JsonObject.

property-name is case-sensitive.

A JsonError is raised If:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a JSON string
- The property value cannot be parsed a Base-64 encoded string

GetTableInfoByID( ) method

Returns information about a temp-table when the ID is specified. The ID is a numeric value that identifies the temp-table and is maintained by the ABL virtual machine (AVM). If the ID is valid for an existing temp-table, this method returns TRUE. Otherwise, it returns FALSE.

Return type: LOGICAL

Access: PUBLIC STATIC

Applies to: Progress.Database.TempTableInfo class

Syntax

This method supports two overloaded versions. The first version returns the temp-table name and name of the procedure or class that instantiated the temp-table.

```
GetTableInfoByID( INPUT temp-table-id AS INTEGER,
                  OUTPUT table-name AS CHARACTER,
                  OUTPUT proc-name AS CHARACTER )
```

The second version returns the handle to the temp-table and the name of the procedure or class that instantiated the temp-table.

```
GetTableInfoByID( INPUT temp-table-id AS INTEGER,
                  OUTPUT table-handle AS HANDLE,
                  OUTPUT proc-name AS CHARACTER )
```

temp-table-id

An integer expression which evaluates the ID of the temp-table.

table-name

Returns a CHARACTER string with the name of the temp-table.

table-handle

Returns a HANDLE to the temp-table whose ID is specified.

proc-name

Returns a CHARACTER string with the procedure (or class) name. For a static temp-table, it is the procedure with the `DEFINE TEMP-TABLE` statement. For a dynamic temp-table, it is the procedure

which executes the TEMP-TABLE-PREPARE() method. The name of the procedure is the name that is specified when you run mydir/test.p.

GetTableInfoByPosition( ) method

Returns information about the position of a temp-table from a list of temp-tables. This is based on the number returned by the TempTableCount property. The method raises an error if it is a not a valid position (not within the TempTableCount range).

Return type: VOID

Access: PUBLIC STATIC

Applies to: Progress.Database.TempTableInfo class

Syntax

This method supports two overloaded versions. The first version returns the temp-table name and name of the procedure or class that instantiated the temp-table.

```
GetTableInfoByPosition( INPUT element AS INTEGER,
                        OUTPUT table-name AS CHARACTER,
                        OUTPUT proc-name AS CHARACTER )
```

The second version returns the handle to the temp-table and the name of the procedure or class that instantiated the temp-table.

```
GetTableInfoByPosition( INPUT element AS INTEGER,
                        OUTPUT table-handle AS HANDLE,
                        OUTPUT proc-name AS CHARACTER )
```

element

An integer expression which evaluates to the position of a given temp-table from a list of temp-tables. It must be a value between 1 and TempTableCount.

table-name

Returns a CHARACTER string with the name of the temp-table.

table-handle

Returns the handle to the temp-table specified by the element parameter.

proc-name

Returns a CHARACTER string with the procedure (or class) name. For a static temp-table, it is the procedure with the DEFINE TEMP-TABLE statement. For a dynamic temp-table, it is the procedure which executes the TEMP-TABLE-PREPARE method. The name of the procedure is the name that is specified when you run mydir/test.p.

The following example shows the temp-tables that are currently in the scope for monitoring deployed clients.

```
DEFINE VARIABLE i AS INTEGER NO-UNDO.
DEFINE VARIABLE hTable AS HANDLE NO-UNDO.
DEFINE VARIABLE cProc AS CHARACTER NO-UNDO.

REPEAT i =1 TO  Progress.Database.TempTableInfo:TempTableCount:
Progress.Database.TempTableInfo:GetTableInfoByPosition (i, OUTPUT hTable, OUTPUT cProc).
DISPLAY hTable:name hTable:DYNAMIC cProc.
END.
```

GetTableStatHistoryHandle( ) method

Returns the handle of the temp-table used for storing the archived statistics from the `_TableStat` virtual system table. The method returns the Unknown value (?) if the `ArchiveTableStatistics` property is set to FALSE.

Return type: HANDLE

Access: PUBLIC STATIC

Applies to: Progress.Database.TempTableInfo class

Syntax

```
GetTableStatHistoryHandle()
```

This method returns a table handle to the `_TableStatHistory` temp-table. This is maintained by the AVM and cannot be deleted when the `ArchiveTableStatistics` property is set to TRUE. If you try to delete it, the AVM raises an error. The `_TableStatHistory` dynamic temp-table can be deleted if the `ArchiveTableStatistics` property is set to FALSE.

The `_TableStatHistory` only contains information about temp-tables that were deleted during the session.

The records in this table can be manipulated by emptying the temp-table. If the `ArchiveTableStatistics` property is again set to TRUE, the dynamic temp-table `_TableStatHistory` is recreated. It contains all the fields from `_TableStat`, except the `_TableStat-id` field.

The fields that will be present in the `_Tablestat` table are listed below:

`_Table-name`

A character field with the temp-table name.

`_Proc-name`

A character field with the name of the procedure in which the temp-table was created.

`_Delete-Timestamp`

The date-time field with the timestamp denoting when the temp-table was deleted.

GetterAccessMode property

Returns an instance of `Progress.Reflect.AccessMode` that indicates the access mode for the GET accessor of the property. If the property has a GET accessor without an implementation, `GetterAccessMode` returns the access mode of the property itself. If the property has no defined GET accessor, `GetterAccessMode` returns the Unknown value (?).

Data type: Progress.Reflect.AccessMode enumeration

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Property class

GetType( ) method (JsonArray)

Gets JSON data type for the identified element. The returned integer corresponds to the static integer properties defined in the Progress.Json.ObjectModel.JsonDataType class.

Return type: INTEGER

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
GetType( INPUT index AS INTEGER )
```

index

An INTEGER identifying the element whose type is to be returned. Indexing into JsonArrays is 1-based.

A JsonError is raised if *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?).

GetType( ) method (JsonObject)

Gets JSON data type for the named property. The returned integer corresponds to the static integer properties defined in the Progress.Json.ObjectModel.JsonDataType class.

Return type: INTEGER

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
GetType( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression naming the property to get the JSON data type.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist

GetType( ) method (TypeHelper)
(Windows only; GUI for .NET only)

Gets a System.Type instance for a .NET class using a fully-qualified class name. Use this method when you must get the run-time type of an object for which you have no instance and cannot instantiate an instance.

If the type cannot be found, this method returns the Unknown value (?) and optionally throws a .NET Exception.

Return type: System.Type class (from the .NET Framework)

Access: PUBLIC STATIC

Applies to: Progress.Util.TypeHelper class

Syntax

```
TypeHelper:GetType ( INPUT class-name AS CHARACTER
                [ , INPUT throw-on-error AS LOGICAL ] )
```

class-name

A CHARACTER express that specifies the fully-qualified name of a class.

throw-on-error

A LOGICAL expression indicating whether to throw an exception if the type cannot be found. If TRUE, the AVM throws an exception. If FALSE, the AVM will not throw an exception.

The following example gets a System.Type instance for the System.Windows.Forms.Button class:

```
USING System.Windows.Forms.* FROM ASSEMBLY.
USING Progress.Util.* FROM ASSEMBLY.

DEFINE VARIABLE btnType AS System.Type NO-UNDO.

btnType = TypeHelper:GetType( "System.Windows.Forms.Button" ).
```

GetVariable( ) method

Returns a `Progress.Reflect.Variable` instance describing the class variable that matches the specified conditions. If there is no matching variable, the method returns the Unknown value (?).

Return type: Progress.Reflect.Variable class

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
GetVariable ( INPUT variable-name AS CHARACTER )

GetVariable ( INPUT variable-name AS CHARACTER,
              INPUT flags AS Progress.Reflect.Flags )
```

variable-name

A CHARACTER expression that evaluates to the name of the variable.

flags

A `Progress.Reflect.Flags` instance indicating the access mode(s), scope(s), and/or class level(s) the returned variable must have. See Progress.Reflect.Flags enumeration for more information.

Notes

- `GetVariable(`*variable-name*`)` only returns public, non-static variables. `GetVariable(`*variable-name, flags*`)` returns all ABL class variables based on the flags specified. However, you can only get or set variables according to the same rules as when reflection is not used. For example, you can only get or set a protected variable from within the class that defines it, or from a subclass.
- This method does not return .NET class fields defined as private.

GetVariables( ) method

Returns an array of `Progress.Reflect.Variable` instances describing the class variables that match the specified conditions. There is no guaranteed order for the returned variables. If there are no variables that match the specified conditions, the method returns an indeterminate array.

Return type: Progress.Reflect.Variable class EXTENT

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
GetVariables ( )

GetVariables ( INPUT flags AS Progress.Reflect.Flags )
```

flags

A `Progress.Reflect.Flags` instance indicating the access mode(s), scope(s), and/or class level(s) the returned variables must have. See Progress.Reflect.Flags enumeration for more information.

Notes

- `GetVariables( )` only returns public, non-static variables. `GetVariables(`*`flags`*`)` returns all ABL class variables based on the flags specified. However, you can only get or set variables according to the same rules as when reflection is not used. For example, you can only get or set a protected variable from within the class that defines it, or from a subclass.
- This method does not return .NET class fields defined as private or internal.

GetValue( ) method

Returns the underlying numeric value of the enumeration member.

Return type: INT64

Access: PUBLIC FINAL

Applies to: Progress.Lang.Enum class

Syntax

```
GetValue ( )
```

For example, the code shown below compares a variable to the underlying values of the `MAX` and `MIN` members:

```
ENUM Range:
     DEFINE ENUM MIN = 255
                 MAX = 2147483647.
END NUM.
```

```
DEFINE VARIABLE ix AS INT64 NO-UNDO.

IF ix < Range:MIN:GetValue() OR ix > Range:MAX:GetValue() THEN ...
```

Notes

- For a flag enum instance that represents multiple flags set, the method returns the single numeric value that represents that combination of flags.

See also

ENUM statement

GetVSTHandle( ) method

Returns the handle of a valid VST temp-table.

Return type: HANDLE

Access: PUBLIC STATIC

Applies to: Progress.Database.TempTableInfo class

Syntax

```
GetVSTHandle( INPUT VST-ID AS INTEGER )
```

VST-ID

An integer expression which evaluates to a valid VST ID. The method raises an error if an invalid value is specified for the ID.

Example:

```
USING Progress.Database.*.

DEFINE VARIABLE hVst AS HANDLE NO-UNDO.

hVst = TempTableInfo:GetVSTHandle(VSTTableIds:ACTIOTYPE).
```

The `GetVSTHandle()` method creates a dynamic temp-table for the VST and returns the handle to the caller. Subsequent calls to the method with the same VST return the same temp-table handle. You cannot call the CLEAR() method on the temp-table handles that you obtain with `GetVSTHandle()` method. You cannot set the TRACKING CHANGES attribute on the buffer of the VST temp-table handles. Any method on the temp-table that is used to create, update, or delete a record in a temp-table VST will fail.

Handle property
(Windows only; GUI for .NET only)

The handle to the ABL data source object to which the BindingSource object is bound. This can be a query, ProDataSet, or buffer object handle.

Data type: HANDLE

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

You can use this property to associate an ABL data source object with an unbound BindingSource object instance at run time.

HandleRequest( ) method (IWebHandler)

An implementation of this method must perform the following actions for an HTTP request:

- Create an instance of `Progress.Web.WebRequest` and process the request.
- Create and populate an instance of `Progress.Web.WebResponse`.
- Create an instance of `Progress.Web.WebReponseWriter` and write the response.

Return type: INTEGER

Access: PUBLIC

Applies to: Progress.Web.IWebHandler interface

Syntax

```
HandleRequest ( )
```

Notes

- The return value of this method is currently unused.

Has( ) method (JsonObject)

Returns a LOGICAL indicating if the JsonObject contains the named property.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
Has( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression naming the property to be tested.

property-name is case-sensitive.

If this parameter is the empty string ("") or is the Unknown value (?), a JsonError is raised.

HasStatics( ) method

Returns TRUE if a given class (or any of its super classes) has any static members. This method supports the reflection capabilities of the Progress.Lang.Class class.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
HasStatics ( )
```

HasWidgetPool( ) method

Returns TRUE if a given class has a widget pool. This method supports the reflection capabilities of the Progress.Lang.Class class.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
HasWidgetPool ( )
```

IgnoreComments property

The `IgnoreComments` property when set to TRUE, causes the parser to ignore comments in JSON strings. Comments appear as either C style (`/*...*/`) or C++ style (`//...`). Setting this property to FALSE causes the parser to fail when it encounters a JSON comment.

Data type: BOOLEAN

Access: PUBLIC

Applies to: Progress.Json.JsonParser class

Note: While setting this property to TRUE allows comments to be parsed, the `Progress.Json.ObjectModel` package has no support for writing JSON comments.

IndexStatId property

The `IndexStatId` property identifies the `IndexStat` VST. This VST contains statistics on the number of accesses to a specific range of indexes. You can pass the `IndexStatId` property to the `GetVSTHandle()` method to return the handle to the `IndexStat` VST. This property is read-only.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

InputValue property
(Windows only; GUI for .NET only)

Returns the input value of the specified field in the current row of the top-level table displayed in the bound .NET control.

You can use this property when the BindingSource object is bound to a query or a buffer. When bound to a ProDataSet object, this property applies only to the top-level table displayed in the .NET control.

Data type: The data type of the corresponding buffer field.

Access: PUBLIC Read-only

Applies to: Progress.Data.BindingSource class

Syntax

```
InputValue[ field-index | field-name ]
```

field-index

An INTEGER expression that evaluates to the 0-based index position of the field in the display.

field-name

A CHARACTER expression that evaluates to the name of the field in the display.

See also

ChildInputValue property, Progress.Data.InputValue class

InvalidConfiguration property

An authentication status code that indicates authentication failure due to a missing or invalid OpenEdge domain configuration.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

Invoke( ) method (Class)

Dynamically invokes a method whose name and parameters must be set at run time. The `Invoke( )` method is one of several reflection methods of the `Progress.Lang.Class` class that provide type information about a class or interface at run time.

Return type: Any data type

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

Four overloaded versions are supported. The first version is used to invoke a method that does not take any parameters.

```
[ return-value = ] class-reference:Invoke (
  INPUT object-reference AS Progress.Lang.Object ,
  INPUT method-name AS CHARACTER )
```

The second overloaded version is used to invoke a method that takes zero or more parameters. Since *parameterlist-object* can be built with zero or more parameters, this version of the Invoke( ) method can be used to invoke any method, even those that do not take any arguments.

```
[ return-value = ] class-reference:Invoke (
  INPUT object-reference AS Progress.Lang.Object ,
  INPUT method-name AS CHARACTER ,
  INPUT parameterlist-object AS Progress.Lang.ParameterList )
```

The third overloaded version is used when invoking a static method that does not take any parameters.

```
[ return-value = ] class-reference:Invoke (
 INPUT method-name AS CHARACTER )
```

The fourth overloaded version is used when invoking a static method that takes zero or more parameters. Since *parameterlist-object* can be built with zero or more parameters, this version of the Invoke( ) method can be used to invoke any static method, even those that do not take any arguments.

```
[ return-value = ] class-reference:Invoke (
  INPUT method-name AS CHARACTER ,
  INPUT parameterlist-object AS Progress.Lang.ParameterList)
```

return-value

An optional data element which is assigned the return value from the invoked, non-void method. The AVM checks *return-value* at run time for data type compatibility with what is actually returned by the method.

class-reference

An object reference to the Progress.Lang.Class class instance containing the type information.

object-reference

A reference to the object that contains the method you want to invoke, and whose type is compatible with the *class-reference* object. When invoking a static method, the Unknown value (?) is passed for this parameter.

method-name

A CHARACTER expression that evaluates at run time to the name of the method to be invoked.

parameterlist-object

An instance of the Progress.Lang.ParameterList class.

Note: The DYNAMIC-INVOKE function provides similar functionality to the Invoke( ) method. The advantage of the former is that it has a fixed, compile-time parameter list and does not require the creation of a Progress.Lang.ParameterList object at run time.

See also

DYNAMIC-INVOKE function, DYNAMIC-NEW statement, New( ) method, NumParameters property (ParameterList), Progress.Lang.ParameterList class, SetParameter( ) method (Class)

Invoke( ) method (Constructor)

Invokes the constructor represented by the `Progress.Reflect.Constructor` instance and creates a new instance of the class. This method can only be used to invoke a public constructor. If used on a non-public constructor, the method raises an error.

Return type: Any class type

Access: PUBLIC

Applies to: Progress.Reflect.Constructor class

Syntax

```
Invoke ( INPUT params AS Progress.Lang.ParameterList )
```

params

A `Progress.Lang.ParameterList` instance containing the parameters for the constructor. If the constructor has no parameters, pass a `ParameterList` instance with its `NumParameters` property

set to 0. If the `ParameterList` instance does not match the parameters for the constructor, the method raises an error.

Invoke( ) method (Method)

Invokes the method represented by the `Progress.Reflect.Method` instance. This method can only be used to invoke a public method. If used on a non-public method, the method raises an error.

Return type: The type, if any, returned by the method

Access: PUBLIC

Applies to: Progress.Reflect.Method class

Syntax

```
Invoke ( INPUT params AS Progress.Lang.ParameterList )

Invoke ( INPUT instance AS Progress.Lang.Object,
         INPUT params AS Progress.Lang.ParameterList )
```

instance

An instance of the class that contains the method. This instance must be of the type specified by either the `DeclaringClass` or `OriginatingClass` property of the `Progress.Reflect.Method` instance. Using the *instance* parameter with a static method will raise an error.

params

A `Progress.Lang.ParameterList` instance containing the parameters for the method. If the method has no parameters, pass a `ParameterList` instance with its `NumParameters` property set to 0. If the `ParameterList` instance does not match the parameters for the method, the method will raise an error.

Notes

- Use the `Invoke( params )` overload to invoke a static method and the `Invoke(instance, params)` an instance method. Calling an overload on a method of the incorrect scope will raise an error.

IsA( ) method

Returns TRUE if a class or interface represented by the `Progress.Lang.Class` object or object type-name expression passed to the method is in the hierarchy of the Progress.Lang.Class class instance on which the method is called. Otherwise, the method returns FALSE. The `IsA( )` method supports the reflection capabilities of the `Progress.Lang.Class` class.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
IsA ( { INPUT object-reference AS Progress.Lang.Class |
        INPUT object-type-name AS CHARACTER } )
```

object-reference

An object reference to a `Progress.Lang.Class` instance that can represent a class or interface type.

object-type-name

A CHARACTER expression that evaluates to a class or interface type name.

This method returns TRUE when the Progress.Lang.Class class instance on which the method is called represents a class type that:

- Is the same class represented by the `Progress.Lang.Class` object or class type-name expression passed to the method.
- Derives from (is a subclass of) the class represented by the `Progress.Lang.Class` object or class type-name expression passed to the method.
- Implements the interface represented by the `Progress.Lang.Class` object or interface type-name expression passed to the method. The passed interface type can be a super interface in the hierarchy of an interface that is implemented by the class type represented by the `Progress.Lang.Class` class on which the method is called.

In this example, the `IsA( )` method is used to determine if a "`Customer`" object is within the hierarchy of a "`Bar`" class. Note that "`Customer`" can represent either a class type or an interface type.

```
DEFINE VARIABLE b        AS LOGICAL             NO-UNDO.
DEFINE VARIABLE barType  AS Progress.Lang.Class NO-UNDO.
DEFINE VARIABLE custType AS Progress.Lang.Class NO-UNDO.

/* Determines if a "Customer" class object is within the hierarchy of
   a "Bar" class */
ASSIGN
  barType  = Progress.Lang.Class:GetClass("Bar")
  custType = Progress.Lang.Class:GetClass("Customer").

b = barType:IsA(custType).
/* or */
b = barType:IsA("Customer").
```

IsAbstract( ) method

Returns a LOGICAL value indicating if the specified object is a class type defined as ABSTRACT. This method supports the reflection capabilities of the Progress.Lang.Class class.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
IsAbstract( )
```

See also

CLASS statement

IsAbstract property

Returns TRUE if the method, property, or event is abstract. Otherwise, it returns FALSE.

Data type: LOGICAL

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Event class, Progress.Reflect.Method class, Progress.Reflect.Property class

IsEnum( ) method

Returns TRUE if the object is an enumeration type. Otherwise, it returns FALSE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
IsEnum( )
```

IsFinal( ) method

Returns a LOGICAL value indicating if the specified object is a class type defined as FINAL.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
IsFinal( )
```

IsFinal property

Returns TRUE if the method is final. Otherwise, it returns FALSE.

Data type: LOGICAL

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Method class

IsFlagsEnum( ) method

Returns TRUE if the object is a flag enumeration type. Otherwise, it returns FALSE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
IsFlagsEnum( )
```

IsFlagSet( ) method

Indicates whether the enum instance has the specified flag(s) set.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.FlagsEnum class

Syntax

```
IsFlagSet ( INPUT flags AS Progress.Lang.FlagsEnum )
```

flags

A reference to a `Progress.Lang.FlagsEnum` instance with one or more flags set. A runtime error will be generated if the underlying value of *flag* is 0. See the Notes for more information.

This method, invoked on *thisInstance*, is equivalent to evaluating the following expression:

(*thisInstance* AND *flags* = *flags*)

This means that if *flags* has multiple flags set, the method returns TRUE only if all the flags in *flags* are set in the enum instance. For example:

```
ENUM Permission:
     DEFINE ENUM None = 0
                 Read
                 Write
                 ReadWrite = Read,Write
                 Create
                 Delete.
END ENUM.
```

```
DEFINE VARIABLE myPermission AS Permission NO-UNDO.
DEFINE VARIABLE lRret        AS LOGICAL    NO-UNDO.

myPermission = Permission:Read.

/* Returns TRUE. */
lRet = myPermission:IsFlagSet(Permission:Read).

/* Returns FALSE because Write is not set in myPermission. */
lRet = myPermission:IsFlagSet(Permission:ReadWrite).

/* Returns FALSE because the method is passed both
   the Read and Write flags because of the bitwise OR,
   but Write is not set in myPermission. */
lRet = myPermission:IsFlagSet(Permission:Read OR Permission:Write).
```

Notes

- The method raises error if the two enum instances are not the same type.
- Because `IsFlagSet()` cannot be passed an enum instance with an underlying value of 0, you must check for that case before using the method, using the EQ (=) or NE (<>) operators or the `Equals()` method, for example. Using the `Permission` enum from above, the following example checks that the user's permission level, `myPermission`, matches the permission level needed, `neededPermissions`, to perform an operation. The code checks to make sure `neededPermissions` is not `Permission:None` first, before invoking `IsFlagSet()`:

  ```
  DEFINE VARIABLE neededPermissions AS Permission NO-UNDO.
  DEFINE VARIABLE myPermission AS Permission NO-UNDO.

  ...

  myPermission = Permission:Read.

  IF (neededPermissions NE Permission:None) OR
     (NOT myPermission:IsFlagSet(neededPermissions)) THEN
      MESSAGE "You do not have permission for this operation.".
  END.
  ```

See also

ENUM statement

IsGeneric( ) method

Returns a LOGICAL value indicating if the specified class is a generic class. This method only returns TRUE for a generic .NET class; it always returns FALSE for ABL classes.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
IsGeneric( )
```

IsGreater( ) method
(Windows only; GUI for .NET only)

Compares the underlying values of the specified enumeration types and returns TRUE if the first value is greater than the second value. Otherwise, it returns FALSE.

Return type: LOGICAL

Access: PUBLIC STATIC

Applies to: Progress.Util.EnumHelper class

Syntax

```
EnumHelper:IsGreater ( INPUT enum1 AS CLASS System.Enum,
                       INPUT enum2 AS CLASS System.Enum )
```

enum1

Static reference to an enumeration value, or a reference to an enumeration type instance.

enum2

Static reference to an enumeration value, or a reference to an enumeration type instance.

Specify a static reference to an enumeration value using the following syntax:

```
enum-type:enum-member
```

Where *enum-type* is the type name of the enumeration and *enum-member* is a member name of the enumeration. For example:

```
System.Windows.Forms.AnchorStyles:Bottom
```

For information on the System.Enum class and .NET enumeration types, refer to the .NET Framework class library.

See also

IsLess( ) method

IsGreaterOrEqual( ) method
(Windows only; GUI for .NET only)

Compares the underlying values of the specified enumeration types and returns TRUE if the first value is greater than or equal to the second value. Otherwise, it returns FALSE.

Return type: LOGICAL

Access: PUBLIC STATIC

Applies to: Progress.Util.EnumHelper class

Syntax

```
EnumHelper:IsGreaterOrEqual ( INPUT enum1 AS CLASS System.Enum,
                              INPUT enum2 AS CLASS System.Enum )
```

enum1

Static reference to an enumeration value, or a reference to an enumeration type instance.

enum2

Static reference to an enumeration value, or a reference to an enumeration type instance.

Specify a static reference to an enumeration value using the following syntax:

```
enum-type:enum-member
```

Where *enum-type* is the type name of the enumeration and *enum-member* is a member name of the enumeration. For example:

```
System.Windows.Forms.AnchorStyles:Bottom
```

For information on the System.Enum class and .NET enumeration types, refer to the .NET Framework class library.

See also

IsLessOrEqual( ) method

IsIndexed( ) method

Returns a LOGICAL value indicating if the specified class is an indexed class. This method only returns TRUE for an indexed .NET class; it always returns FALSE for ABL classes.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
IsIndexed( )
```

IsIndexed property

Returns TRUE for indexed properties of .NET classes. For ABL class properties and non-indexed .NET class properties, it returns FALSE.

Data type: LOGICAL

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Property class

IsInterface( ) method

If the specified object is defined as an interface type, this method returns TRUE. Otherwise, it returns FALSE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
IsInterface( )
```

IsLess( ) method
(Windows only; GUI for .NET only)

Compares the underlying values of the specified enumeration types and returns TRUE if the first value is less than the second value. Otherwise, it returns FALSE.

Return type: LOGICAL

Access: PUBLIC STATIC

Applies to: Progress.Util.EnumHelper class

Syntax

```
EnumHelper:IsLess ( INPUT enum1 AS CLASS System.Enum,
                    INPUT enum2 AS CLASS System.Enum )
```

enum1

Static reference to an enumeration value, or a reference to an enumeration type instance.

enum2

Static reference to an enumeration value, or a reference to an enumeration type instance.

Specify a static reference to an enumeration value using the following syntax:

```
enum-type:enum-member
```

Where *enum-type* is the type name of the enumeration and *enum-member* is a member name of the enumeration. For example:

```
System.Windows.Forms.AnchorStyles:Bottom
```

For information on the System.Enum class and .NET enumeration types, refer to the .NET Framework class library.

See also

IsGreater( ) method

IsLessOrEqual( ) method
(Windows only; GUI for .NET only)

Compares the underlying values of the specified enumeration types and returns TRUE if the first value is less than or equal to the second value. Otherwise, it returns FALSE.

Return type: LOGICAL

Access: PUBLIC STATIC

Applies to: Progress.Util.EnumHelper class

Syntax

```
EnumHelper:IsLessOrEqual ( INPUT enum1 AS CLASS System.Enum,
                           INPUT enum2 AS CLASS System.Enum )
```

enum1

Static reference to an enumeration value, or a reference to an enumeration type instance.

enum2

Static reference to an enumeration value, or a reference to an enumeration type instance.

Specify a static reference to an enumeration value using the following syntax:

```
enum-type:enum-member
```

Where *enum-type* is the type name of the enumeration and *enum-member* is a member name of the enumeration. For example:

```
System.Windows.Forms.AnchorStyles:Bottom
```

For information on the `System.Enum` class and .NET enumeration types, refer to the .NET Framework class library.

See also

IsGreaterOrEqual( ) method

IsNull( ) method (JsonArray)

Returns a LOGICAL indicating if an element in a JsonArray contains the null value.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
IsNull( INPUT index AS INTEGER )
```

index

An INTEGER identifying the element to be tested. Indexing into JsonArrays is 1-based.

A JsonError is raised if *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?).

IsNull( ) method (JsonObject)

Returns a LOGICAL indicating if a property in a JsonObject contains the null value.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
IsNull(INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression naming the property to be tested.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist

IsOverride property

Returns TRUE if the event, method, or property is an override. Otherwise, it returns FALSE.

Data type: LOGICAL

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Event class, Progress.Reflect.Method class, Progress.Reflect.Property class

IsSerializable( ) method

Indicates whether the specified object is of a class type that is defined as SERIALIZABLE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

```
IsSerializable( )
```

IsStatic property

Returns TRUE if the event, method, variable, or property is static. Otherwise, it returns FALSE.

Data type: LOGICAL

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Event class, Progress.Reflect.Method class, Progress.Reflect.Variable class

Length property (MemoryOutputStream)

The initial size of the `Data` property when the memory stream was initialized. If the number of bytes written is greater than the initial size of the`Data` property, then `Length` returns the number of bytes written. The value of `Length` will always be equivalent to `GET-SIZE( MemoryOutputStream:Data)`.

Data type: INT64

Access: PUBLIC Read-only

Applies to: Progress.IO.MemoryOutputStream class

Length property (JSON)

Indicates the current length of the array. This value can be both read and set. If it is set to a smaller value, the array is truncated. Any element beyond the new smaller size of the array is treated as if you call the `JsonArray:Remove( )` on them. If this value is set to a larger size, the array is extended with null values as if `JsonArray:AddNull( )` was called.

Data type: INTEGER

Access: PUBLIC Readable/Writeable

Applies to: Progress.Json.ObjectModel.JsonArray class

Load( ) method (Class)
(Windows only; GUI for .NET only)

Loads resources from a .NET XML resource (`.resx`) file associated with a .NET form class, and returns a `System.Resources.ResXResourceSet` instance through which you can access the resources by name.

Return type: `System.Resources.ResXResourceSet` class (from the .NET Framework)

Access: PUBLIC STATIC

Applies to: Progress.Util.ResourceHelper class

Syntax

```
ResourceHelper:Load ( INPUT filename AS CHARACTER, INPUT path AS CHARACTER )
```

filename

A CHARACTER expression that specifies a .NET XML resource (`.resx`) file that contains the resources you want to load. You can specify *filename* with an absolute pathname or a relative pathname based on *path*. If you do not specify *path*, the AVM looks for the file in the current working directory.

A resource file is always named the same as the form with a `.resx` extension, and is located in the same directory as the source code for the form.

path

A CHARACTER expression that specifies a relative path in which to search for the file specified in *filename*. This is typically "PROPATH" or a subset of your PROPATH. If you do not specify a path, you must specify the empty string ("").

The following example loads the resource file `form1.resx` from PROPATH and accesses an image in the resource file:

```
USING System.Windows.Forms.* FROM ASSEMBLY.
USING Progress.Util.*        FROM ASSEMBLY.

DEFINE VARIABLE resources AS System.Resources.ResXResourceSet NO-UNDO.
DEFINE VARIABLE image     AS System.Drawing.Image             NO-UNDO.

resources = ResourceHelper:Load( "form1.resx", "PROPATH" ).
image = CAST( resources:GetObject( "open.Image" ), System.Drawing.Image ).
```

For information on the System.Resources.ResXResourceSet class, refer to the .NET Framework class library.

LoginDenied property

An authentication status code that indicates authentication failed because of a denied login attempt.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

LoginLockout property

An authentication status code that indicates authentication failed because of a login failure lockout (e.g., too many failed attempts).

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

MapNullToEmptyString property
(Windows only; GUI for .NET only)

Indicates whether the `BindingSource` returns the Unknown (?) value or an empty string when the value in the .NET object is null. If `MapNullToEmptyString` is set to TRUE, then the empty string is returned; if set to FALSE, then the Unknown (?) value is returned.

Data type: LOGICAL

Access: PUBLIC Readable/Writable

Applies to: Progress.Data.BindingSource class

MaxDataGuess property
(Windows only; GUI for .NET only)

An estimate of the number of records that a query will return. The property's default value is zero.

Data type: INTEGER

Access: PUBLIC Readable/Writable

Applies to: Progress.Data.BindingSource class

The MaxDataGuess property offers a time-saving alternative for a ProBindingSource that binds to a query with a large result set. Some .NET controls must know the number of records in the result set when they initialize. By default, the ProBindingSource counts the records by reading the records in the query's result set, which can be time-consuming for large result sets. Setting this property provides the .NET control with an estimated value to bypass the counting process.

Because the .NET control used an estimated number when it visualized, you might see it readjust as the ProBindingSource's Count property develops a more accurate count of the records. For example, you might notice the vertical scrollbar thumb of a grid changing size to more accurately reflect the number of displayed records.

When using this property, remember the following:

- Set MaxDataGuess before opening the query.
- Set MaxDataGuess before setting the ProBindingSource's Handle property.
- The MaxDataGuess and Batching properties cannot both be TRUE at the same time. The ProBindingSource throws an exception if this happens.
- Do not open the query with the PRESELECT option. This is an exception to the normal rule with ProBindingSource queries. Since a PRESELECT query builds the entire result list, it negates the optimization from MaxDataGuess.

The following code snippet shows the correct order for using the MaxDataGuess property:

```
DEFINE VARIABLE rBindS AS Progress.Data.BindingSource NO-UNDO.
DEFINE VARIABLE hQuery AS HANDLE                      NO-UNDO.

ASSIGN
  rBindS              = NEW Progress.Data.BindingSource()
  rBindS:MaxDataGuess = 200000.

CREATE QUERY hQuery.
hQuery:SET-BUFFERS(BUFFER Customer:HANDLE).
hQuery:QUERY-PREPARE("FOR EACH Customer NO-LOCK").
hQuery:QUERY-OPEN().

rBindS:Handle = hQuery.
```

See also

Count property

MaxTries property

An authentication status code that indicates failure because the limit on the number of login retries was exceeded.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

MissingCredentials property

An authentication status code that indicates failure because one or more of the required user credentials is missing from the CLIENT-PRINCIPAL.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

Mode property

Returns an instance of `Progress.Reflect.ParameterMode` that indicates the access mode for the parameter (e.g., `Input`, `Output`, etc).

Data type: Progress.Reflect.ParameterMode enumeration

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Parameter class

Note: For parameters with a `DataType` property value of `Buffer`, this property is always `Input`.

MstrBlkId property

The `MstrBlkId` property identifies the `MstrBlk` VST. This VST contains before-image and after-image logging information, and two-phase commit status. You can pass the `MstrBlkId` property to the `GetVSTHandle()` method to return the handle to the `MstrBlk` VST. This property is read-only.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

Name property (Reflect)

Returns a CHARACTER expression holding the name of the constructor, data member, event, method, parameter, or property.

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Constructor class, Progress.Reflect.Event class, Progress.Reflect.Method class, Progress.Reflect.Parameter class, Progress.Reflect.Variable class

New( ) method

Instantiates a class whose class type and any parameters are known only at run time. This method returns a `Progress.Lang.Object`, which provides an instance of a class. The `New( )` method provides similar functionality to the DYNAMIC-NEW function. The advantage to the latter is that it has a fixed, compile-time parameter list and does not require the creation of a `Progress.Lang.ParameterList` object at run time.

Return type: Progress.Lang.Object class

Access: PUBLIC

Applies to: Progress.Lang.Class class

The class type to instantiate is specified by the TypeName property of the applicable `Progress.Lang.Class` instance.

Syntax

Two overloaded versions are supported. The first version is used when the constructor of the class does not take any arguments.

```
New ( )
```

You can use the second overloaded version of the New( ) method when the constructor of a class takes zero or more parameters. Since a *parameterlist-object* can be built with zero or more parameters, this version of the New( ) method can be used to instantiate any class, even those classes whose constructor does not take any arguments.

```
New ( INPUT parameterlist-object AS Progress.Lang.ParameterList )
```

parameterlist-object

An instance of the Progress.Lang.ParameterList class. The instance contains an entry for each parameter required by the constructor being invoked. It is valid to pass a Progress.Lang.ParameterList object with zero parameters if the class constructor does not take any parameters. If one or more parameters are passed, the Progress.Lang.ParameterList:SetParameter( ) method must have been previously called on each parameter.

The AVM raises a run-time error if the Unknown value (?) is passed for *parameterlist-object*.

Notes

- Since the New( ) method returns an instance of the Progress.Lang.Object class, the result of the New( ) method must be assigned to a Progress.Lang.Object object.
- The New( ) method must be coded in a standalone Assignment (=) statement and not as part of an ASSIGN statement.
- For information on any restrictions, such as types of .NET objects you can instantiate, see the reference entry for the DYNAMIC-NEW statement.

See also

Clear( ) method (Class), DYNAMIC-INVOKE function, DYNAMIC-NEW statement, Invoke( ) method (Class), NEW function (classes), NEW statement, NumParameters property (ParameterList), Progress.Lang.ParameterList class, SetParameter( ) method (Class)

NewRow property
(Windows only; GUI for .NET only)

Indicates whether the current row in the bound .NET control is a new row.

Data type: LOGICAL

Access: PUBLIC Read-only

Applies to: Progress.Data.BindingSource class

When TRUE, the row that currently has focus is a newly created row that can still be undone. The row remains in a new state until you either move focus to another row or discard the new row. When FALSE, the current row is not a new row.

NextForm property
(Windows only; GUI for .NET only)

The next .NET `Progress.Windows.IForm` or ABL window `FormProxy` (`Progress.Windows.FormProxy`) object in the list of all valid OpenEdge form objects created in the current ABL session.

Data type: Progress.Windows.IForm interface

Access: PUBLIC Read-only

Applies to: Progress.Windows.Form class, Progress.Windows.FormProxy class, Progress.Windows.IForm interface

The chain of forms begins with SESSION:FIRST-FORM and ends with SESSION:LAST-FORM. Once your position in the chain of OpenEdge forms is established, you can use the `NextForm` and `PrevForm` properties to walk the chain.

Note: If you instantiate a `Progress.Windows.IForm` in a non-GUI ABL session, the object does not appear in the chain of OpenEdge forms, but only in the list of valid class-based objects, which you can access using the FIRST-OBJECT attribute or the LAST-OBJECT attribute.

See also

FIRST-FORM attribute, LAST-FORM attribute, PrevForm property

NEXT-SIBLING property

The object reference for the next ABL or .NET class instance in the list of instances created in the current ABL session. The value of this property is available after obtaining a valid object reference (for example, by using the SESSION:FIRST-OBJECT attribute to obtain the object reference for the first class instance in the list). If there are no class object instances in the current session, or you have gone past the last class instance in the list, this attribute returns the Unknown value (?).

Data type: Progress.Lang.Object class

Access: PUBLIC Read-only

Applies to: Progress.Lang.Object class

Once your position in the list is established, you can use the NEXT-SIBLING and PREV-SIBLING properties to walk the list of class object instances.

There are two points worth noting about this property with garbage collection enabled:

- The object references maintained by the NEXT-SIBLING property and the FIRST-OBJECT attribute do not count as references for garbage collection. That is, if a class instance is referenced only on the session object chain, it is available for automatic garbage collection.
- When an ABL object that inherits from a .NET object is on the session object chain, it can have circular references that prevent it from being deleted (as with pure ABL objects). When there are no more ABL references to such an ABL-derived .NET object, the object is removed from the object chain, but may still be instantiated because of references to it from .NET. If the application re-obtains a reference to the object in ABL by getting it from .NET, the reference is again placed on the object chain.

To check the validity of an object reference, use the VALID-OBJECT function.

See also

PREV-SIBLING property

Null property

Returned by the GetType method on a JsonObject or JsonArray indicating that the JSON data type for the specified property or element is null.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Json.ObjectModel.JsonDataType class

Number property

Returned by the GetType method on a JsonObject or JsonArray indicating that the JSON data type for the specified property or element is number.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Json.ObjectModel.JsonDataType class

NumHandlerParameters property

Returns the number of parameters defined by the event handler.

Data type: INTEGER

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Event class

NumMessages property

This property indicates how many error number and error message pairs are currently available in the error object.

Data type: INTEGER

Access: PUBLIC Read-only

Applies to: Progress.Lang.ProError class, System.Exception class (OpenEdge-enhanced)

In ABL, an error is represented as a pair of values including a message number and an error message. The message number is a unique number identifying the particular error. The error message is a string which describes the error.

For a .NET Exception object, the number of error number and error message pairs includes messages from any InnerException objects.

See also

GetMessage( ) method (Class), GetMessageNum( ) method

NumParameters property (ParameterList)

Identifies the number of parameters in a Progress.Lang.ParameterList object.

Data type: INTEGER

Access: PUBLIC Readable/Writable

Applies to: Progress.Lang.ParameterList class

This property can be updated, which has the same effect as the Clear( ) method (Class) of the Progress.Lang.ParameterList class in that all information is released. This is true even if the new number of parameters is the same as the previous number. The new list of parameters is then initialized with the new number of parameters. Each parameter returns to the non-set condition and cannot be used until a Progress.Lang.ParameterList:SetParameter( ) method is run on it.

See also

Clear( ) method (Class), New( ) method, NumParameters property (ParameterList), Progress.Lang.ParameterList class, SetParameter( ) method (Class)

NumParameters property (Reflect)

Returns the number of parameters of the constructor or method.

Data type: INTEGER

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Constructor class, Progress.Reflect.Method class

Object property

Returned by the GetType method on a JsonObject or JsonArray indicating that the JSON data type for the specified property or element is a JSON Object.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Json.ObjectModel.JsonDataType class

OEClientType property

Contains a string value that identifies the type of client/server. The property contains a comma separated list of different values. The value and the client type associated with it are listed in the following table.

Table 142: Client type and value information

Value	Client Type
`"4GLCLIENT"`	ABL Client
`"JAVACLIENT"`	Java Open Client
`".NETCLIENT"`	.NET Open Client
`"MULTI-SESSION-AGENT"`	Progress Application Server agent

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Lang.OEVersionInfo class

OEMaintVersion property

Contains the maintenance version number for OpenEdge. For the first release of OpenEdge 11.0, the value of this property is `"0"`.

Note: The value of `OEMaintVersion` property is the Unknown value (?) if you are running OpenEdge versions prior to 11.0.

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Lang.OEVersionInfo class

OEMajorVersion property

Contains the major version number of OpenEdge. For OpenEdge 11.0, the value of this property is `"11"`.

Note: The value of `OEMajorVersion` property is the Unknown value (?) if you are running OpenEdge versions prior to 11.0.

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Lang.OEVersionInfo class

OEMinorVersion property

Contains the minor version number of OpenEdge. For OpenEdge Release 11.0, the value of this property is `"0"`.

Note: The value of `OEMinorVersion` property is the Unknown value (?) if you are running OpenEdge versions prior to 11.0.

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Lang.OEVersionInfo class

Offset property

Indicates the offset in bytes, in the JSON String at which the error occurs.

Data type: INTEGER

Access: PUBLIC

Applies to: Progress.Json.JsonParserError class

Or( ) method
(Windows only; GUI for .NET only)

Performs a bitwise inclusive OR operation on the underlying values of the specified enumeration types and returns a new instance. You typically use this method to set (turn on) a flag.

Return type: System.Enum class (from the .NET Framework)

Access: PUBLIC STATIC

Applies to: Progress.Util.EnumHelper class

Syntax

```
EnumHelper:Or ( INPUT enum1 AS CLASS System.Enum,
                INPUT enum2 AS CLASS System.Enum )
```

enum1

Static reference to an enumeration value, or a reference to an enumeration type instance.

enum2

Static reference to an enumeration value, or a reference to an enumeration type instance.

Specify a static reference to an enumeration value using the following syntax:

```
enum-type:enum-member
```

Where *enum-type* is the type name of the enumeration and *enum-member* is a member name of the enumeration. For example:

```
System.Windows.Forms.AnchorStyles:Bottom
```

The following example uses this method to set a flag:

```
USING System.Windows.Forms.* FROM ASSEMBLY.
USING Progress.Util.* FROM ASSEMBLY.

DEFINE VARIABLE rMyStyle AS System.Enum.

/* Adds Bottom and Right to the list */
rMyStyle = EnumHelper:Or( AnchorStyles:Bottom, AnchorStyles:Right ).
DISPLAY STRING( rMyStyle ) FORMAT "x(20)" SKIP.
/* Displays "Bottom, Right" */

/* Adds Top to the list */
rMyStyle = EnumHelper:Or( rMyStyle, AnchorStyles:Top ).
DISPLAY STRING( rMyStyle ) FORMAT "x(20)".
/* Displays "Top, Bottom, Right" */.
```

If you want to use the returned System.Enum instance as a different enumeration type, you must cast the instance to that specific type using the CAST function.

For information on the System.Enum class and .NET enumeration types, refer to the .NET Framework class library.

See also

CAST function, Xor( ) method

OriginatingClass property

Returns a `Progress.Lang.Class` instance representing the class that the `Progress.Reflect.Constructor`, `Progress.Reflect.Event`, `Progress.Reflect.Method`, or `Progress.Reflect.Variable` object was retrieved from.

Data type: Progress.Lang.Class class

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Constructor class, Progress.Reflect.Event class, Progress.Reflect.Method class, Progress.Reflect.Variable class

Note that the DeclaringClass property does not necessarily match the OrignatingClass property. For example, consider the classes defined in the following code fragment:

```
CLASS ClassA:
  METHOD PUBLIC VOID MethodA:
    MESSAGE "A method has been called." VIEW-AS ALERT-BOX.
  END METHOD.
END CLASS.

CLASS ClassB INHERITS ClassA:
END CLASS.

CLASS ClassC INHERITS ClassB:
  METHOD PUBLIC OVERRIDE VOID MethodA:
    MESSAGE "An overriden method has been called." VIEW-AS ALERT-BOX.
  END METHOD.
END CLASS.
```

Calling GetMethod("MethodA", params) on instances of Progress.Lang.Class derived from each of the three classes would return Progress.Reflect.Method instances with the DeclaringClass properties set as shown in the table:

Progress.Lang.Class instance	Class indicated by the DeclaringClass property	Class indicated by the OriginatingClass property
ClassA	ClassA	ClassA
ClassB	ClassA	ClassB
ClassC	ClassC	ClassC

The DeclaringClass property for ClassB returns ClassA because MethodA is defined in ClassA and not overidden by ClassB, whereas the OriginatingClass property returns ClassB. The OriginatingClass property returns the name of the class associated with the Progress.Lang.Class instance that you obtained the Progress.Reflect.Method instance from.

See also: DeclaringClass property

Package property

The package portion of the specified class or interface type name. If the class or interface type name does not contain a package, the value of this property is the Unknown value (?).

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Lang.Class class

See also

Type-name syntax

Parse( ) method

Parses the JSON found from the specified source and returns a JsonConstruct object when the parse is complete. The returned value is either a JsonObject or a JsonArray. If the JSON syntax is incorrect, a JsonParserError is raised.

Return type: Progress.Json.ObjectModel.JsonConstruct class

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.ObjectModelParser class

Syntax

```
Parse( INPUT web-context-stream AS HANDLE )

Parse( INPUT source AS LONGCHAR )

Parse( INPUT source AS MEMPTR )

Parse( INPUT source AS CHARACTER )
```

web-context-stream

A HANDLE variable that specifies the WEB-CONTEXT system handle.

source

An ABL variable or expression containing the JSON string in memory. A LONGCHAR must have a Unicode code page ("UTF-8", "UTF-16", "UTF-16BE", "UTF-16LE", "UTF-32", "UTF-32BE" or "UTF-32LE"). The code page for the data in a MEMPTR is detected by looking at the JSON to be parsed. If the code page is not one of the valid Unicode code pages, a JsonError is raised.

ParseFile( ) method

Parses the JSON found from the specified file and returns a JsonConstruct object when the parse is complete. The returned value is either a JsonObject or a JsonArray. If the JSON syntax is incorrect, a JsonParserError is raised.

Return type: Progress.Json.ObjectModel.JsonConstruct class

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.ObjectModelParser class

Syntax

```
ParseFile( INPUT file-name AS CHARACTER )
```

file-name

A CHARACTER expression naming a file system location containing the JSON string. You can specify an absolute path or a path relative to the current working directory. If the file does not exist or cannot be opened, an error is raised.

PasswordExpired property

An authentication status code that indicates authentication failed because the user account password has expired.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

PermissionDenied property

An authentication status code that indicates failure because user does not have the necessary privileges to access the user account information required to complete an authentication or SSO operation.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

Position property (BindingSource)
(Windows only; GUI for .NET only)

The zero-based position (index) of the current row in the bound .NET control.

Data type: INTEGER

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

Whenever the value of this property changes based on some user action in the .NET control or programatically, the BindingSource object automatically synchronizes the buffer in the bound ABL data source object to correspond to the selected row in the bound .NET control.

Note: Progress Software Corporation recommends that you do not navigate or reposition the query associated with the bound ABL data source object. For more information about synchronizing data, see *Use GUI for .NET Objects in ABL Applications*.

See also

PositionChanged event

Position property (MemoryInputStream)

The number of bytes currently read from a `MemoryInputStream` object.

Data type: INT64

Access: PUBLIC Read-only

Applies to: Progress.IO.MemoryInputStream class

Position property (Parameter)

Returns an integer representing the one-based position of the parameter in the parameter list of the constructor, event handler, or method.

Data type: INTEGER

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Parameter class

PrevForm property
(Windows only; GUI for .NET only)

The previous .NET `Progress.Windows.IForm` or ABL window `FormProxy` (`Progress.Windows.FormProxy`) object in the list of all valid OpenEdge form objects created in the current ABL session.

Data type: Progress.Windows.IForm interface

Access: PUBLIC Read-only

Applies to: Progress.Windows.Form class, Progress.Windows.FormProxy class, Progress.Windows.IForm interface

The chain of forms begins with SESSION:FIRST-FORM and ends with SESSION:LAST-FORM. Once your position in the chain of OpenEdge forms is established, you can use the `NextForm` and `PrevForm` properties to walk the chain.

Note: If you instantiate a `Progress.Windows.IForm` in a non-GUI ABL session, the object does not appear in the chain of OpenEdge forms, but only in the list of valid class-based objects, which you can access using the FIRST-OBJECT attribute or the LAST-OBJECT attribute.

See also

FIRST-FORM attribute, LAST-FORM attribute, NextForm property

PREV-SIBLING property

The object reference for the previous ABL or .NET class instance in the list of instances created in the current ABL session. The value of this property is available after obtaining a valid object reference (for example, by using the SESSION:LAST-OBJECT attribute to obtain the object reference for the last class instance in the list). If there are no class object instances in the current session, or you have gone past the first class instance in the list, this attribute returns the Unknown value (?).

Data type: Progress.Lang.Object class

Access: PUBLIC Read-only

Applies to: Progress.Lang.Object class

Once your position in the list is established, you can use the NEXT-SIBLING and PREV-SIBLING properties to walk the list of class object instances.

There are two points worth noting about this property with garbage collection enabled:

- The object references maintained by the PREV-SIBLING property and the LAST-OBJECT attribute do not count as references for garbage collection. That is, if a class instance is referenced only on the session object chain, it is available for automatic garbage collection.
- When an ABL object that inherits from a .NET object is on the session object chain, it can have circular references that prevent it from being deleted (as with pure ABL objects). When there are no more ABL references to such an ABL-derived .NET object, the object is removed from the object chain, but may still be instantiated because of references to it from .NET. If the application re-obtains a reference to the object in ABL by getting it from .NET, the reference is again placed on the object chain.

To check the validity of an object reference, use the VALID-OBJECT function.

See also

NEXT-SIBLING property

ProcedureName property

The filename or pathname of the remote procedure that an application server client invoked for the request executing in the current application server session.

Note: This property has no meaning in the context of a non-application server ABL session.

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Lang.OERequestInfo class

The value returned by this property depends on how the remote procedure is invoked on the client. If the client calls the procedure using an absolute or relative path, the same absolute or relative pathname is returned. If the client calls the procedure with no path, only the filename of the remote procedure is returned.

You can query this property in any procedure or class running in an application server session. This includes any application server configuration procedures that run as part of a remote procedure request, such as the Activate and Deactivate procedures that run for each request to an application server running in session-free operating mode.

An application server session can access this property using the CURRENT-REQUEST-INFO attribute on the SESSION system handle.

See also

CURRENT-REQUEST-INFO attribute

ProWinHandle property
(Windows only; GUI for .NET only)

The handle to one of the following ABL window widgets:

- The shadow window associated with a .NET form that is used for parenting relationships between ABL windows and .NET forms. ABL creates a shadow window for every .NET form that you instantiate from the Progress.Windows.Form class.
- An ABL window that is accessible from its associated Progress.Windows.FormProxy object. ABL creates an instance of this object for every ABL window that you create in a session that accesses .NET forms.

This property is used when mixing ABL windows and .NET forms in a single application.

Data type: HANDLE

Access: PUBLIC Read-only

Applies to: Progress.Windows.Form class, Progress.Windows.FormProxy class, Progress.Windows.IForm interface

The following example shows a class that creates and displays a .NET form and an ABL window, and parents the form to the window:

```
USING System.Windows.Forms.* FROM ASSEMBLY.
USING Progress.Windows.* FROM ASSEMBLY.

DEFINE VARIABLE win1  AS HANDLE NO-UNDO.
DEFINE VARIABLE form1 AS Progress.Windows.Form NO-UNDO.

/* Create a form */
form1      = NEW Progress.Windows.Form().
form1:TEXT = "Form1".
form1:Show().

/* Create a window */
CREATE WINDOW win1 ASSIGN
  TITLE        = "Win1"
  HEIGHT-CHARS = 5
  WIDTH-CHARS  = 35
  VISIBLE      = TRUE
  SENSITIVE    = TRUE.

/* Make the form the parent of the window */
win1:PARENT = form1:ProWinHandle.

WAIT-FOR Application:RUN( form1 ).

DELETE WIDGET win1.
```

For more information on using ABL windows along with .NET forms in your application, see *Use GUI for .NET Objects in ABL Applications*.

Publish( ) method

Publishes an ABL class event defined in the current class definition. This is a built-in event method for managing class events. For more information, see the Publish( ) event method reference entry.

Read( ) method (InputStream)

This method reads data from an input stream and stores the information in the specified location or variable. It returns the total number of characters read for CHARACTER and LONGCHAR data targets, or the total number of bytes read for a MEMPTR target.

Return type: INT64

Access: PUBLIC

Applies to: Progress.IO.InputStream class

Syntax

```
ABSTRACT Read( INPUT target AS MEMPTR,
               INPUT offset AS INT64,
               INPUT length AS INT64 )

Read( INPUT target AS MEMPTR )

ABSTRACT Read( INPUT delimiter AS CHARACTER,
               OUTPUT target AS CHARACTER )

ABSTRACT Read( INPUT delimiter AS CHARACTER,
               OUTPUT target AS LONGCHAR )
```

delimiter

When specified, the method reads from the current input stream until it encounters the value in this CHARACTER variable.

length

Indicates how many bytes the method should read from the current input stream.

offset

Indicates which byte of the *target* MEMPTR the method should start writing to.

target

A CHARACTER, LONGCHAR, or MEMPTR variable that the method uses to store the data it reads.

Notes

- All of the overloads of this method are abstract except for `Read( INPUT target AS MEMPTR )`. The behavior described for the parameters of the abstract methods are provided as guidelines.

- The method reads data from the file input stream until the MEMPTR is filled, the number of bytes specified by *length* is read, or the end of the stream is reached, whichever happens first.

Read( ) method (JsonArray)

Uses the contents of a temp-table to generate a JsonArray. The current contents of the JsonArray are removed.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
Read( INPUT TABLE-HANDLE tt-handle )

Read( INPUT TABLE-HANDLE tt-handle,
      INPUT omit-initial-values AS LOGICAL )
```

tt-handle

Represents a handle to either a static temp-table or dynamic temp-table to be used as the source for setting the JsonArray. Each element in this JsonArray is a JsonObject representing a row in the temp-table. If this value is the Unknown value (?), a JsonError is raised.

omit-initial-values

An LOGICAL expression where:

- TRUE directs the AVM to exclude temp-table fields containing their initial values from the JSON object.
- FALSE directs the AVM to include all temp-table field data in the JSON. The default value of the expression is FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

When working with large temp-tables, omitting fields containing their initial values can yield smaller JSON values, more efficient network transfers, and performance gains with the Read( ) method.

This behavior applies both to temp-table fields which have the default initial value for its data type, and for fields which have an initial value set with the ABL INITIAL option.

The ABL READ-JSON( ) method always populates created records with initial values from the temp-table or ProDataSet definition. Other applications might not do this.

Notes

- When executing the method, there is no guarantee as to the contents of the buffers involved after the method finishes executing.
- Binary types, including BLOB, RAW, and ROWID are base64-encoded/decoded. See Data type mapping in *Use JSON with ABL Applications* for more information.

Example

When a temp-table handle is passed to `JsonArray:Read( )`, the `JsonArray` is set to a series of objects representing each row in the temp table. The following is a temp-table definition that is passed to `JsonArray:Read( )`:

```
DEFINE VARIABLE httCust AS HANDLE NO-UNDO.
DEFINE VARIABLE myArr AS JsonArray NO-UNDO.
DEFINE VARIABLE myLongchar AS LONGCHAR NO-UNDO.

DEFINE TEMP-TABLE ttCust
  FIELD CustNum AS INTEGER
  FIELD Name    AS CHARACTER
  FIELD NewCust AS LOGICAL.

/*
 .
 . Load temp-tables
 .
 */

httCust = TEMP-TABLE ttCust:HANDLE.
myArr = NEW JsonArray().
myArr:Read(TABLE-HANDLE httCust).
myArr:Write(myLongchar, TRUE).
```

The following is the resultant JSON construct tree, `myLongchar`:

```
[
  {"CustNum": 1, "Name": "Lift Tours", "NewCust": false },
  {"CustNum": 2, "Name": "FlyByNight", "NewCust": true }
]
```

Read( ) method (JsonObject)

Uses the data from a ProDataSet, temp-table, or buffer handle to generate a `JsonObject`. The current contents of the `JsonObject` are removed.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
Read( INPUT DATASET-HANDLE pds-handle )

Read( INPUT DATASET-HANDLE pds-handle,
      INPUT omit-initial-values AS LOGICAL )

Read( INPUT DATASET-HANDLE pds-handle,
      INPUT omit-initial-values AS LOGICAL,
      INPUT read-before-image AS LOGICAL)

Read( INPUT TABLE-HANDLE tt-handle )

Read( INPUT TABLE-HANDLE tt-handle,
      INPUT omit-initial-values AS LOGICAL )

Read( INPUT buffer-handle AS HANDLE )

Read( INPUT buffer-handle AS HANDLE,
      INPUT omit-initial-values AS LOGICAL )
```

pds-handle

A HANDLE to either a static ProDataSet or dynamic ProDataSet to be used as the source for setting the `JsonObject`. If this value is the Unknown value (?), a `JsonError` is raised.

tt-handle

A HANDLE to either a static temp-table or dynamic temp-table to be used as the source for setting the JsonObject. If this value is the Unknown value (?) a `JsonError` is raised.

buffer-handle

A HANDLE to either a static temp-table buffer or dynamic temp-table buffer to be used as the source for setting the `JsonObject`. If this value is the Unknown value (?), a `JsonError` is raised.

Note: The value of the `JsonObject` is based on the one row in the buffer, not the entire table. This behavior differs from `WRITE-JSON( )` and `WRITE-XML( )`. Those methods process the entire table associated with the buffer.

---

omit-initial-values

A LOGICAL expression where TRUE directs the AVM to exclude temp-table fields containing their initial values from the JSON object, and FALSE directs the AVM to include all temp-table field data in the JSON. The default value is FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

When working with large ProDataSets, omitting fields containing their initial values can yield smaller JSON values, more efficient network transfers, and performance gains with the `JsonObject( )` constructor.

This behavior applies both to temp-table fields that have the default initial value for its data type, and for fields that have an initial value set with the ABL `INITIAL` option. When a temp-table handle is passed to `JsonObject:Read( )`, the `JsonObject` is set to a single property.

read-before-image

A LOGICAL expression where TRUE directs the AVM to include ProDataSet before-image data and error information in the JSON object. The default value is FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

Notes

- You cannot execute the method on a database buffer.
- When executing the method, there is no guarantee as to the contents of the buffers involved after the method finishes executing.
- Binary types, including BLOB, RAW, and ROWID are base64-encoded/decoded. See Data type mapping in *Use JSON with ABL Applications* for more information.

Example

The following example shows a temp-table definition passed to `JsonObject:Read( ):`

```
DEFINE VARIABLE httCust AS HANDLE NO-UNDO.
DEFINE VARIABLE myObj AS JsonObject NO-UNDO.
DEFINE VARIABLE myLongchar AS LONGCHAR NO-UNDO.

DEFINE TEMP-TABLE ttCust
  FIELD CustNum AS INTEGER
  FIELD Name    AS CHARACTER
  FIELD NewCust AS LOGICAL.

/*
 .
 . Load temp-table
 .
 */

httCust = TEMP-TABLE ttCust:HANDLE.
myObj = NEW JsonObject().
myObj:Read(TABLE-HANDLE httCust).
myObj:Write(myLongchar, TRUE).
```

The following is the resultant JSON construct tree, `myLongchar`:

```
{"ttCust":
  [
    {"CustNum": 1, "Name": "Lift Tours", "NewCust": false },
    {"CustNum": 2, "Name": "FlyByNight", "NewCust": true }
  ]
}
```

Read( ) method (MemoryInputStream)

Reads from a `MemoryInputStream` object into a CHARACTER, LONGCHAR, or MEMPTR and returns the number of bytes read.

Return type: INT64

Access: PUBLIC

Applies to: Progress.IO.MemoryInputStream class

Syntax

```
Read(INPUT pmData AS MEMPTR)
Read(INPUT pmData AS MEMPTR, INPUT offset AS INT64, INPUT len AS INT64)
Read(INPUT delimiter AS CHARACTER, OUTPUT lcData AS LONGCHAR)
Read(INPUT delimiter AS CHARACTER, OUTPUT cData AS CHARACTER)
```

pmData

Specifies a MEMPTR where the data read from a `MemoryInputStream` object is written until the MEMPTR is filled or until the end of the stream is reached (whichever comes first).

offset

Offset, in bytes, to the first byte in the MEMPTR.

len

The number of bytes to read from a `MemoryInputStream` object.

delimiter

A single character that indicates where a read from a `MemoryInputStream` object should stop. Note that:

- If you specify a string of more than one character, only the first character is used.
- If you specify a null string (`""`), a space character is used as the delimiter.
- If you specify an alphabetic character, the delimiter is case sensitive.

lcData

The name of a LONGCHAR variable where the data read from a `MemoryInputStream` object is placed. The AVM returns a runtime error if the total number of bytes read, before a delimiter is encountered, cannot fit into a LONGCHAR expression.

cData

The name of a CHARACTER variable where the data read from a `MemoryInputStream` object is placed. The AVM returns a runtime error if the total number of bytes read, before a delimiter is encountered, cannot fit into a CHARACTER expression.

ReadByte( ) method

Read and return an unsigned 1-byte value from the current `MemoryInputStream` object.

Note: This method is equivalent to the `GET-BYTE` function, which applies to MEMPTR variables but not to `MemoryInputStream` objects.

Return type: INTEGER

Access: PUBLIC

Applies to: Progress.IO.MemoryInputStream class

Syntax

```
ReadByte( )
```

ReadDouble( ) method

Read and return a signed double value from the current `MemoryInputStream` object.

Note: This method is equivalent to the `GET-DOUBLE` function, which applies to MEMPTR variables but not to `MemoryInputStream` objects.

Return type: DECIMAL

Access: PUBLIC

Applies to: Progress.IO.MemoryInputStream class

Syntax

```
ReadDouble( )
```

ReadFloat( ) method

Read and return a signed floating-point value from the current `MemoryInputStream` object.

Note: This method is equivalent to the `GET-FLOAT` function, which applies to MEMPTR variables but not to `MemoryInputStream` objects.

Return type: DECIMAL

Access: PUBLIC

Applies to: Progress.IO.MemoryInputStream class

Syntax

```
ReadFloat( )
```

ReadInt64( ) method

Read and return a signed 8-byte value from the current `MemoryInputStream` object.

Note: This method is equivalent to the `GET-INT64` function, which applies to MEMPTR variables but not to `MemoryInputStream` objects.

Return type: INT64

Access: PUBLIC

Applies to: Progress.IO.MemoryInputStream class

Syntax

```
ReadInt64( )
```

ReadLong( ) method

Read and return a signed 4-byte value from the current `MemoryInputStream` object.

Note: This method is equivalent to the `GET-LONG` function, which applies to MEMPTR variables but not to `MemoryInputStream` objects.

Return type: INT64

Access: PUBLIC

Applies to: Progress.IO.MemoryInputStream class

Syntax

```
ReadLong( )
```

ReadLongchar( ) method

Using the null terminator as a delimiter, reads the next set of characters up to the null terminator from the input stream and returns the set of characters read as a LONGCHAR expression. The AVM returns a runtime error if the total number of bytes read, before the null delimiter is encountered, cannot fit into a LONGCHAR expression.

Return type: LONGCHAR

Access: PUBLIC

Applies to: Progress.IO.MemoryInputStream class

Syntax

```
ReadLongchar( )
```

ReadShort( ) method

Read and return a signed 2-byte value from the current `MemoryInputStream` object.

Note: This method is equivalent to the `GET-SHORT` function, which applies to MEMPTR variables but not to `MemoryInputStream` objects.

Return type: INTEGER

Access: PUBLIC

Applies to: Progress.IO.MemoryInputStream class

Syntax

```
ReadShort( )
```

ReadString( ) method

Using the null terminator as a delimiter, read the next set of characters up to the null terminator from the input stream and return the set of characters read as a CHARACTER expression. The AVM returns a runtime error if the total number of bytes read, before the null terminator is encountered, cannot fit into a CHARACTER expression.

Return type:CHARACTER

Access: PUBLIC

Applies to: Progress.IO.MemoryInputStream class

Syntax

```
ReadString( )
```

ReadUnsignedLong( ) method

Read and return an unsigned 4-byte value from the current `MemoryInputStream` object.

Note: This method is equivalent to the `GET-UNSIGNED-LONG` function, which applies to MEMPTR variables but not to `MemoryInputStream` objects.

Return type: INT64

Access: PUBLIC

Applies to: Progress.IO.MemoryInputStream class

Syntax

```
ReadUnsignedLong( )
```

ReadUnsignedShort( ) method

Read and return an unsigned 2-byte value from the current `MemoryInputStream` object.

Note: This method is equivalent to the `GET-UNSIGNED-SHORT` function, which applies to MEMPTR variables but not to `MemoryInputStream` objects.

Return type: INTEGER

Access: PUBLIC

Applies to: Progress.IO.MemoryInputStream class

Syntax

```
ReadUnsignedShort( )
```

ReadOnly property

This property is TRUE if the value of the associated `DataSlot` can be read but not set; otherwise, the property is FALSE.

Data type: LOGICAL

Access: PUBLIC Read-only

Refresh( ) method (Progress.Data.BindingSource)
(Windows only; GUI for .NET only)

Refreshes field values displayed for the current or specified row in any bound .NET control with values from the corresponding record in the bound ABL data source object.

You can use this method when the BindingSource object is bound to a query or a buffer. When bound to a ProDataSet object, this method applies only to the top-level table displayed in the .NET control.

Note: To refresh a row in a child table, you must use the RefreshAll( ) method.

Return type: VOID

Access: PUBLIC

Applies to: Progress.Data.BindingSource class

Syntax

```
Refresh ( [ INPUT record-index AS INTEGER ] )
```

record-index

An optional INTEGER value specifying a 1-based record index position in either the result set of a bound query or the top-level query in a ProDataSet object. If specified, the BindingSource object refreshes the corresponding row in any bound .NET control with values from the specified record in the bound ABL data source object. If not specified, the BindingSource object refreshes the current row in the bound .NET control (specified by the Position property) with values from the corresponding record in the bound ABL data source object.

Invoking this method does not change the editing mode of the .NET control.

Note: If you refill the temp-table whose data you are viewing, you need to reopen the query you are bound to (or the TOP-NAV-QUERY of the dataset if you are bound to a dataset) before you call Refresh( ) or RefreshAll( ). The BindingSource works off of a query result set. If the result set is not correct, Refresh( ) or RefreshAll( ) do not help. Similarly, if you add or delete records, you must either add/remove entries from the result set, or reopen the appropriate query. For more information, see the Synchronize data section in *Use GUI for .NET Objects in ABL Applications*.

See also

AutoSync property, Position property (BindingSource), RefreshAll( ) method

RefreshAll( ) method
(Windows only; GUI for .NET only)

Refreshes field values displayed for all rows (parent and child) in any bound .NET control with values from the corresponding records in the bound ABL data source object.

You can use this method when the BindingSource object is bound to a query or a buffer. When bound to a ProDataSet object, this method applies to all levels.

Note: To refresh a single row in a query, buffer, or top-level table, use the Refresh( ) method.

Return type: VOID

Access: PUBLIC

Applies to: Progress.Data.BindingSource class

Syntax

```
RefreshAll( )
```

Invoking this method does not change the editing mode of the .NET control.

Note: If you refill the temp-table whose data you are viewing, you need to reopen the query you are bound to (or the `TOP-NAV-QUERY` of the dataset if you are bound to a dataset) before you call `Refresh( )` or `RefreshAll( )`. The `BindingSource` works off of a query result set. If the result set is not correct, `Refresh( )` or `RefreshAll( )` do not help. Similarly, if you add or delete records, you must either add/remove entries from the result set, or reopen the appropriate query. For more information, see the Synchronize data section in *Use GUI for .NET Objects in ABL Applications*.

See also

AutoSync property

Remove( ) method (JsonArray)

Removes one or more elements from the JsonArray. All the remaining elements are shifted up and JsonArray:Length is decremented by the number of elements removed.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
Remove( INPUT index AS INTEGER )

Remove( INPUT index AS INTEGER,
        INPUT count AS INTEGER )
```

index

Indicates an INTEGER identifying the element to be removed. Indexing into JsonArray is 1-based.

count

An INTEGER parameter that specifies the number of elements, starting with the element at *index*, to remove.

A JsonError is raised if:

- *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?)
- *count* is less than 1 or is the Unknown value (?)

- The sum of *index* and *count* is greater than the length of the JsonArray

Remove( ) method (JsonObject)

Removes the named property from the JsonObject. On successful execution, this method returns TRUE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
Remove( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression naming the property to be removed from the object.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist

RemoveMessage( ) method

Removes the indexed error (both error message string and error message number) from the message list. The method decrements the NumMessages property by 1 and moves the messages after the indexed error forward in the list by 1.

For example, if there are five messages in the AppError object's list of messages, and the RemoveMessage method has a *MessageIndex* of 3, then the method removes message three. It also updates message four to be message three, and message five to be message four. Finally, it sets NumMessages to 4.

Return type: VOID

Access: PUBLIC

Applies to: Progress.Lang.AppError class

Syntax

```
RemoveMessage ( INPUT MessageIndex AS INTEGER )
```

RequestId property

Contains a unique value that identifies the client request currently running in an application server session. The client generates a value for the `RequestId` property and is passed on to the application server session as a part of each request. The initial value for this property is a randomly generated character string containing a globally unique identifier.

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Lang.OERequestInfo class

You can access this property in an:

- **Application server session** — Using the CURRENT-REQUEST-INFO and CURRENT-RESPONSE-INFO attributes on the SESSION system handle.
- **ABL client session** — Using the REQUEST-INFO and RESPONSE-INFO attributes on the server object handle used to invoke the remote request, and on the asynchronous request object handle returned for an asynchronous request.

See also

CURRENT-REQUEST-INFO attribute, CURRENT-RESPONSE-INFO attribute, REQUEST-INFO attribute, RESPONSE-INFO attribute

Reset( ) method (MemoryInputStream)

Set the reading position of the current stream.

Return type: VOID

Access: PUBLIC

Applies to: Progress.IO.MemoryInputStream class

Syntax

```
Reset(INPUT position AS INT64)
```

position

The position in the current stream you want to set the reading position to. If the value of *position* is less than zero or greater than the amount of bytes available to read, the AVM generates a runtime error.

ReturnExtent property

When the method described by the `Progress.Reflect.Method` instance returns an array, `ReturnExtent` returns the size (extent) of the array. If the array is indeterminate in its initial state with no size, `ReturnExtent` returns the Unknown value (?). If the method does not return an array, `ReturnExtent` returns 0.

Data type: INTEGER

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Method class

ReturnType property

Returns an instance of `Progress.Reflect.DataType` that indicates the data type (e.g., `Character`, `Integer`, etc.) of the value returned by the method. If the return type is a class-based object, then the `ReturnTypeName` property returns a class type name, including package, as a CHARACTER expression.

Data type: Progress.Reflect.DataType link t/k

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Method class

See also

ReturnTypeName property

ReturnTypeName property

When the associated `ReturnType` property indicates a class-based object, `ReturnTypeName` returns a class type name, including package, as a CHARACTER expression. If `ReturnType` does not indicate an object, `ReturnTypeName` returns the Unknown value (?).

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Method class

Note: For .NET class types, `ReturnTypeName` returns the class type name used in the ABL, not the assembly-qualified name.

See also

ReturnType property

ReturnValue property

Typically, the place you programatically set the value for a RETURN or RETURN ERROR statement in an error object. The ReturnValue property, when set by the AVM, contains the error string from a RETURN ERROR statement.

If you were not using an error object, these values would usually be accessed through the RETURN-VALUE function. You would also find the values in the RETURN-VALUE function when:

- A program throws an AppError and the caller suppresses the error with the NO-ERROR option.
- The AVM creates an AppError object when encountering a RETURN ERROR *`ErrorString`* statement. *`ErrorString`* is used to populate the ReturnValue property.

Data type: CHARACTER

Access: PUBLIC Readable/Writeable

Applies to: Progress.Lang.AppError class

RowModified property
(Windows only; GUI for .NET only)

Indicates whether the current row in the bound .NET control is currently being edited.

Data type: LOGICAL

Access: PUBLIC Read-only

Applies to: Progress.Data.BindingSource class

When TRUE, the row that currently has focus is a newly modified row that can still be undone. The row remains in a modified state until you either move focus to another row or cancel the modification (by pressing **ESCAPE**, for example). When FALSE, the row has not been modified.

Use this property in an event handler to determine whether any data needs to be validated and saved to the ABL data source object.

RowsAdded property
(Windows only; GUI for .NET only)

Use this property in an OffEnd event handler to indicate the number of records added to the result set from the most recent batch operation.

Data type: INTEGER

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.OffEndEventArgs class

See also

Batching property, OffEnd event (.NET)

Serialize( ) method (BinarySerializer)

Serializes an ABL class-based object to the specified binary stream.

Return type: VOID

Access: PUBLIC

Applies to: Progress.IO.BinarySerializer class

Syntax

```
Serialize ( INPUT object-reference AS Progress.Lang.Object,
            INPUT output-stream AS Progress.IO.OutputStream )
```

object-reference

The class-based object to be serialized.

output-stream

The stream to which the *object-reference* is serialized.

The following code sample serializes and deserializes `myObj`:

```
DEFINE VARIABLE myFileOutStream AS Progress.IO.FileOutputStream.
DEFINE VARIABLE myFileInStream  AS Progress.IO.FileInputStream.
DEFINE VARIABLE mySerializer    AS Progress.IO.BinarySerializer.
DEFINE VARIABLE myObj           AS Acme.MyClass.

myObj = NEW  Acme.MyClass().

mySerializer = NEW Progress.IO.BinarySerializer().

/* Serialize object */
myFileOutStream = NEW Progress.IO.FileOutputStream("MyClass.bin").
mySerializer:Serialize(myObj, myFileOutStream).
myFileOutStream:Close().

/* Deserialize object */
myFileInStream = NEW Progress.IO.FileInputStream("MyClass.bin").
myObj = CAST(mySerializer:Deserialize(myFileInStream), Acme.MyClass).
myFileInStream:Close().
```

Notes

- All non-static data members, properties, temp tables, and ProDataSets of the object are serialized. Object references in the data members of a class are also serialized.
- The following restrictions apply to objects being serialized:
  - The object's class and all of the classes in its hierarchy must be marked as SERIALIZABLE. See the CLASS statement entry for more information.
  - All of the object's data members that are themselves defined as class-based objects must be of class types that are also marked SERIALIZABLE. ABL enums are also allowed because they are implicitly serializable.

  - To be able to deserialize an object, the API for the object's class at the time of serialization must match the API for the class at the time of deserialization. In other words, all the data members, properties, methods signatures and events must match exactly.
  - The content of static data members is not serialized, and the state of queries, buffers, open files, streams, and event subscriptions, for example, are not maintained.
  - Statically defined temp-tables and ProDataSets in user-defined classes are serialized, except for REFERENCE-ONLY tables.

- You can serialize multiple objects to one output stream before calling the `Close( )` method. The objects must be deserialized in the same order as they were serialized.
- If the output stream is unknown or cannot be written to, the AVM raises an error.

Serialize( ) method (JsonSerializer)

Serializes an ABL class-based object to the specified JSON stream.

Return type: VOID

Access: PUBLIC

Applies to: Progress.IO.JsonSerializer class

Syntax

```
Serialize( INPUT object-reference AS Progress.Lang.Object,
           INPUT output-stream AS Progress.IO.OutputStream )
```

object-reference

The class-based object to be serialized.

output-stream

The stream to which the *object-reference* is serialized.

The following code sample serializes and deserializes `myObj`:

```
DEFINE VARIABLE myFileOutStream AS Progress.IO.FileOutputStream.
DEFINE VARIABLE myFileInStream  AS Progress.IO.FileInputStream.
DEFINE VARIABLE mySerializer    AS Progress.IO.JsonSerializer.
DEFINE VARIABLE myObj           AS Acme.MyClass.

myObj = NEW  Acme.MyClass().

mySerializer = NEW Progress.IO.JsonSerializer(FALSE).

/* Serialize object */
myFileOutStream = NEW Progress.IO.FileOutputStream("MyClass.json").
mySerializer:Serialize(myObj, myFileOutStream).
myFileOutStream:Close().

/* Deserialize object */
myFileInStream = NEW Progress.IO.FileInputStream("MyClass.json").
myObj = CAST(mySerializer:Deserialize(myFileInStream), Acme.MyClass).
myFileInStream:Close().
```

The following is an example of a class (`Acme.MyClass`) to be serialized:

```
CLASS Acme.MyClass SERIALIZABLE:
    DEFINE PUBLIC VARIABLE vMyint as INT NO-UNDO.

    DEFINE PUBLIC PROPERTY pMychar AS CHAR
        GET.
        SET.

    DEFINE PROTECTED SERIALIZABLE TEMP-TABLE myTT
        FIELD myChar as CHAR
        FIELD myInt as INT.

    DEFINE PROTECTED SERIALIZABLE DATASET myDset for myTT.

    CONSTRUCTOR PUBLIC MyClass():
        vMyint = 5.
        pMychar = "pMyCharProperty".
        CREATE myTT.
        myTT.myChar = "hello world".
        myTT.myInt = 255.
    END CONSTRUCTOR.
END CLASS.
```

The following is the resulting JSON produced (`MyClass.json`) from calling the `Serialize( )` method:

```
{
  "prods:version": 1,
  "prods:objId": 1,
  "Acme.MyClass": {
    "vMyint": 5,
    "pMychar": "pMyCharProperty",
    "prods:datasets": {
      "myDset": {
        "myTT": [
          {
            "myChar": "hello world",
            "myInt": 255
          }
        ]
      }
    }
```

```
  }
}
```

Notes

- The following restrictions apply to objects being serialized:
  - The object's class and all of the classes in its hierarchy must be marked as SERIALIZABLE. See the CLASS statement entry for more information.
  - All of the object's data members that are themselves defined as class-based objects must be of class types that are also marked SERIALIZABLE. ABL enums are also allowed because they are implicitly serializable.
  - By default, all non-static, public data members and read-write properties are serialized. You can include protected and private data members, read-write properties, temp tables, and ProDataSets in serialization by defining them using the SERIALIZABLE option. See the DEFINE entry for each element for more details.
  - To be able to deserialize an object, the API for the object's class at the time of serialization must be compatible with the API for the class at the time of deserialization. At a minimum, the data members contained in the JSON string for that object should exist in the class, be serializable, and have compatible data types.
  - The content of static data members is not serialized, and the state of queries, buffers, open files, streams, and event subscriptions, for example, are not maintained.
- Unlike binary serialization, you can only serialize one object at a time with JSON. If you need to serialize multiple objects, you must instantiate a serializer for each one.
- If the output stream is unknown or cannot be written to, the AVM raises an error.

SessionId property (OERequestInfo)
(PAS for OE only)

Identifies the Progress Application Server session associated with a given request or response. This property holds the value used by the multi-session agent, not the session manager. If there is no valid response or request running in the session, the property defaults to the Unknown value (?).

Data type: INTEGER

Access: PUBLIC Read-only

Applies to: Progress.Lang.OERequestInfo class

Set( ) method (JsonArray)

Sets an element of the JsonArray to the specified value. On successful completion, the method returns TRUE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
Set( INPUT index AS INTEGER, INPUT value AS CHARACTER )
Set( INPUT index AS INTEGER, INPUT value AS COM-HANDLE )
Set( INPUT index AS INTEGER, INPUT value AS DATE )
Set( INPUT index AS INTEGER, INPUT value AS DATETIME )
Set( INPUT index AS INTEGER, INPUT value AS DATETIME-TZ )
Set( INPUT index AS INTEGER, INPUT value AS DECIMAL )
Set( INPUT index AS INTEGER, INPUT value AS HANDLE )
Set( INPUT index AS INTEGER, INPUT value AS INT64 )
Set( INPUT index AS INTEGER, INPUT value AS INTEGER )
Set( INPUT index AS INTEGER, INPUT value AS LOGICAL )
Set( INPUT index AS INTEGER, INPUT value AS LONGCHAR )
Set( INPUT index AS INTEGER, INPUT value AS MEMPTR )
Set( INPUT index AS INTEGER, INPUT value AS RAW )
Set( INPUT index AS INTEGER, INPUT value AS RECID )
Set( INPUT index AS INTEGER, INPUT value AS ROWID )
Set( INPUT index AS INTEGER, INPUT value AS JsonObject )
Set( INPUT index AS INTEGER, INPUT value AS JsonArray )
```

index

An INTEGER value that indicates the element in the array to be changed. Indexing into JsonArray is 1-based.

A JsonError is raised if *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?).

value

Indicates the value to which the existing element is to be set. The data type of the element is defined by the data type mapping. If *value* is the Unknown value (?), the element is set to the JSON null value.

The data type of the JSON value set by this method call depends upon the ABL data type of the *value* parameter.

Table 143: Value parameter

A value parameter of data type	Data type of JSON value
CHARACTER, LONGCHAR	string
INTEGER, INT64, DECIMAL	number

A value parameter of data type	Data type of JSON value
LOGICAL	boolean
MEMPTR, RAW, ROWID	string with a value as if you had called BASE64-ENCODE( ) on *value*
DATE, DATETIME, DATETIME-TZ	string with a value as if you had called ISO-DATE( ) on *value*
COM-HANDLE, HANDLE, RECID	number with a value as if you had called INTEGER( ) on *value*

Setting an element to an ABL value that maps to a different JSON data type than the current data type changes the elements's data type. The following code sample sets `iType` to JsonDataType:BOOLEAN:

```
idx = myArray:Add("Hello World").
lSuccess = myArray:Set(idx, TRUE).
iType = myArray:GetType(idx).
```

Set( ) method (JsonObject)

Sets the named property of JsonObject to the given value. On successful execution, the method returns TRUE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
Set( INPUT property-name AS CHARACTER,
INPUT value AS CHARACTER )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS COM-HANDLE )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS DATE )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS DATETIME )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS DATETIME-TZ )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS DECIMAL )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS HANDLE )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS INT64 )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS INTEGER )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS LOGICAL )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS LONGCHAR )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS MEMPTR )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS RAW )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS RECID )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS ROWID )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS CLASS JsonObject )

Set( INPUT property-name AS CHARACTER,
     INPUT value AS CLASS JsonArray )
```

property-name

A CHARACTER expression that indicates the name of the existing property whose value is to be changed.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist

value

Indicates the new value to which the property is to be set. The data type of the property is defined by the data type mapping described below. If value is the Unknown value (?) the property is set to a JSON null value.

The data type of the JSON value set by this method call depends upon the ABL data type of the *value* parameter.

Table 144: Value parameter

A value parameter of data type	Data type of JSON value
CHARACTER, LONGCHAR	string
INTEGER, INT64, DECIMAL	number
LOGICAL	boolean
MEMPTR, RAW, ROWID	string with a value as if you had called BASE64-ENCODE( ) on *value*
DATE, DATETIME, DATETIME-TZ	string with a value as if you had called ISO-DATE( ) on *value*
COM-HANDLE, HANDLE, RECID	number with a value as if you had called INTEGER( ) on *value*

Setting a property to an ABL value that maps to a different JSON data type than the current data type changes the elements's data type. For example:

```
lSuccess = myObj:Add("myProp", "Hello World").
lSuccess = myObj:Set("myProp", TRUE).
iType = myObj:GetType("myProp").
```

In the above example, iType is set to Progress.Json.ObjectModel.JsonDataType:BOOLEAN. (See Progress.Json.ObjectModel.JsonDataType class.)

If you set the property to a JsonObject or JsonArray that references this JsonObject or a JSON construct further up the JSON construct tree, a JsonError is raised.

Set( ) method (Reflect)

Sets the value of the data member or property. This method can only be used to set the value of a public data member or property; otherwise, this method will raise an error.

Return type: VOID

Access: PUBLIC

Applies to: Progress.Reflect.Variable class

Syntax

```
Set ( INPUT value AS data-type )

Set ( INPUT index AS INTEGER,
      INPUT value AS data-type )

Set ( INPUT instance AS Progress.Lang.Object,
      INPUT value AS data-type )

Set ( INPUT instance AS Progress.Lang.Object,
      INPUT index AS INTEGER,
      INPUT value AS data-type )
```

data-type

The Progress data type of the value being set. The method will raise an error if the value supplied is not type compatible with the data type of the data member or property.

index

For a data member or property that is an array, the index indicates which element to set. If one of the overloads containing this parameter is invoked for a data member or property that is not an array, the method will raise an error.

instance

An instance of the class that contains the data member or property. This instance must be of the type specified by either the `DeclaringClass` or `OriginatingClass` property of the `Progress.Reflect.Variable` instance. Using the *instance* parameter with a static data member or property will raise an error.

value

The value to set. If the class data member or property being set is an array and you are not setting an individual element of that array, this value will be copied to every element of the array.

Notes

- When applied to a property, the method will invoke the property's SET method, if there is one.
- Use the `Set(`*value*`)` and `Set(`*index*`, `*value*`)` overloads to set the value of static data members, and use the `Set(`*instance*`, `*value*`)` and `Set(`*instance*`, `*index*`, `*value*`)` overloads for instance data members. Calling an overload on a data member of the incorrect scope will raise an error.
- The `Set(`*index*`)` and `Set(`*instance*`, `*index*`)` overloads cannot be used to set individual elements in a .NET array.

SetByteOrder( ) method

Sets the `ByteOrder` property to an indicator that designates the byte-order of MemoryOutputStream object's `Data` property. By default, the byte order for the `Data` property is the `HostByteOrder` member of `Progress.IO.ByteOrder`.

Note: This method is equivalent to the SET-BYTE-ORDER statement, which applies to MEMPTR variables but not to MemoryOutputStream objects.

Return type: VOID

Access: PUBLIC

Applies to: Progress.IO.MemoryOutputStream class

Syntax

```
SetByteOrder(INPUT byte-order AS Progress.IO.ByteOrder)
```

byte-order

Byte order can be specified as PROGRESS.IO.ByteOrder:LittleEndian, PROGRESS.IO.ByteOrder:BigEndian, or PROGRESS.IO.ByteOrder:HostByteOrder.

Notes

- The SetByteOrder( ) method does not affect data currently in the stream. It only has an effect on data written by subsequent calls to one of a MemoryOutputStream's Write methods.
- The Clear( ) method does not reset the stream's byte-order if the byte-order was changed by the SetByteOrder( ) method.

SetClientPrincipal( ) method

Stores a deep copy (clone) of a client principal object in an OERequestInfo instance.

On successful execution this method returns TRUE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.OERequestInfo class

Syntax

```
SetClientPrincipal( INPUT hClientPrincipal AS HANDLE )
```

hClientPrincipal

A handle that refers to a valid client principal object, or to the Unknown value (?). If *hClientPrincipal* is the Unknown value (?), then a client principal object within the OERequestInfo instance is deleted. If *hClientPrincipal* is invalid, the method returns FALSE.

The bold code in the following fragment shows setting a client principal in a server's server's `OERequestInfo` instance:

```
DEFINE VARIABLE hAppSrv      as HANDLE.
DEFINE VARIABLE hCPIn        as HANDLE.
DEFINE VARIABLE hCPOut       as HANDLE.
DEFINE VARIABLE ccid         as CHARACTER.
DEFINE VARIABLE bOK          as LOGICAL.
DEFINE VARIABLE loginState   as CHARACTER.

CREATE SERVER hAppSrv.
ccid = hAppSrv:REQUEST-INFO:clientContextID.

CREATE CLIENT-PRINCIPAL hCPIn.
hCPIn:INITIALIZE("userName@domainName", ccid, ?, "password").
hAppSrv:REQUEST-INFO:SetClientPrincipal(hCPIn).

bOK = hAppSrv:CONNECT("-URL http://hostname:port/apsv").

hCPOut = hAppSrv:RESPONSE-INFO:GetClientPrincipal().
IF VALID-HANDLE(hCPOut)
THEN loginState = hCPOut:LOGIN-STATE.
ELSE loginState = ?.

. . .

IF VALID-HANDLE(hCPOut)
   THEN DELETE OBJECT hCPOut.
```

See also

GetClientPrincipal( ) method (Progress.Lang.OERequestInfo), CURRENT-REQUEST-INFO attribute, CURRENT-RESPONSE-INFO attribute, LOCAL-VERSION-INFO attribute, REQUEST-INFO attribute, RESPONSE-INFO attribute

SetFields( ) method
(Windows only; GUI for .NET only)

Specifies which columns are bound to a ProBindingSource instance, exposing them for bound .NET controls. You use these method overloads before you bind a data source object to a previously unbound ProBindingSource instance.

Note: Do not use this method on a ProBindingSource that is already bound to a data source object. The ProBindingSource throws a .NET exception if this method is used after binding to a data source object.

Return type: VOID

Access: PUBLIC

Applies to: Progress.Data.BindingSource class

Syntax

```
BindingSource:SetFields ( INPUT include-fields     AS CHARACTER ,
                          INPUT except-fields      AS CHARACTER ,
                        {   ""
                        |   INPUT parent-buffer-hdl  AS HANDLE
                        |   INPUT parent-buffer-name AS CHARACTER   } )
```

include-fields

A CHARACTER expression that evaluates to a comma-separated list of fields in the ABL data source object to bind to the ProBindingSource. To include all fields, specify an asterisk (*) in *include-fields*. To include all but a few fields, specify an asterisk (*) in *include-fields* and a comma-separated list of fields to exclude in *except-fields*. If *include-fields* contains specific fields, *except-fields* is ignored.

When including fields for a ProDataSet object, you must specify a comma-separated list of fields to display in display order and qualify the field names with their buffer name. For example:

```
"table1.field1,table1.field2,table2.field1,table1.field3"
```

You must specify at least one field from each table in the hierarchy to be displayed.

When including fields for a query or a join query, you must specify a comma-separated list of fields to display in display order and you should qualify ambiguous field names with their buffer name.

Do not specify a RAW temp-table or database field in *include-fields*. If you explicitly specify a RAW field, the AVM generates a run-time error. If you specify an asterisk ("*") to include all fields and there is a RAW field in the table, the AVM excludes it automatically even if it is not specified in *except-fields*.

except-fields

A CHARACTER expression that evaluates to a comma-separated list of fields in the ABL data source object to exclude from binding to the ProBindingSource. You can specify fields to exclude only when *include-fields* contains an asterisk (*). If you do not have any fields to exclude, you must specify the empty string (""). If *include-fields* contains specific fields, *except-fields* is ignored.

parent-buffer-name

A CHARACTER variable that represents the name of a parent buffer in a ProDataSet object. This corresponds to the top-level table displayed in a hierarchical control (such as a grid or a treeview). If the data source object is not a ProDataSet, this parameter must be the empty string (""). If you specify a name and then bind to another type of data source object, the ProBindingSource throws a .NET exception.

parent-buffer-hdl

A HANDLE variable that represents the handle to a parent buffer in a ProDataSet object. This corresponds to the top-level table displayed in a hierarchical control (such as a grid or a treeview). If the data source object is not a ProDataSet, this parameter must be the empty string (""). If you specify a handle and then bind to another type of data source object, the ProBindingSource throws a .NET exception.

The following code fragment shows the correct sequence for using this method:

1. Create an unbound ProBindingSource instance.
2. Use SetFields( ) to specify the appropriate columns to expose.
3. Set the ProBindingSource's Handle property to bind the data source object.

```
DEFINE VARIABLE pbs AS Progress.Data.BindingSource NO-UNDO.

/* 1 */
pbs = NEW Progress.Data.BindingSource().

/* 2 */
pbs:SetFields("Customer.CustNum,Customer.Name,Customer.State","","").

/* 3 */
pbs:Handle = myQryHdl.
```

See also

Handle property

SetFlag( ) method

Returns a flag enum instance with the indicated flag(s) set.

Return type: Flag enum type

Access: PUBLIC

Applies to: Progress.Reflect.Flags enumeration, Progress.Reflect.ParameterMode enumeration, any user-defined flag enum type

Syntax

```
SetFlag ( INPUT flag AS flag-enum-type )
```

flag

A reference to a flag enum instance. The enum type (*flag-enum-type*) of this instance must match the type of the instance the method is invoked on.

The following example results in `vReflectFlags`, an instance of `Progress.Reflect.Flags`, with the `Public` and `Protected` flags set:

```
DEFINE VARIABLE vReflectFlags AS Flags.
vReflectFlags = Flags:Public.

/* This sets the Protected flag without affecting the status of
   any of the other flags in vAccess. */
vReflectFlags = vReflectFlags:SetFlag(Flags:Protected).
```

You can use a bitwise OR operation to achieve the same result. For example, replacing the last line with `vReflectFlags = vReflectFlags OR Flags:Protected` also sets the `Protected` flag.

Notes

- This method is available for all built-in flag enums, and the compiler automatically generates it for all user-defined flag enums. For example, this code excerpt uses a user-defined `Permission` flag enum, initializes `vPerm` with the `Read` flag set, and then uses `SetFlag( )` to set the `Create` flag, resulting in an enum instance with both the `Read` and `Create` flags set:

```
ENUM Permission FLAGS:
    DEFINE ENUM None = 0
                Read
                Write
                ReadWrite = Read,Write
                Create
                Delete.
END ENUM.
```

```
DEFINE VARIABLE vPerm AS Permission.
vPerm = Permission:Read.

/* This sets the Create flag without affecting the status of
   any of the other flags in vPerm. */
vPerm = vPerm:SetFlag(Permission:Create).
```

See also

OR operator (bitwise), ToggleFlag( ) method, UnsetFlag( ) method

---

SetNull( ) method (JsonArray)

Sets the element at *index* to null for JsonArray.

This method is useful in situations where you know at compile time that are setting an element to the JSON null value. To use one of the `Set( )` overloadings, you must set a variable to the Unknown value (?) first, and then call `Set( )` with the variable. Using the Unknown value (?) as a constant in one of the `Set( )` overloadings will cause a compilation error, because the compiler cannot resolve the overloading to be used.

On successful execution this method returns TRUE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
SetNull( INPUT index AS INTEGER )
```

index

An INTEGER value identifying the element to set to null. Indexing into JsonArray is 1-based.

A JsonError is raised if *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?).

SetNull( ) method (JsonObject)

Set the named property of JsonObject to null.

This method is useful in situations where you know at compile time that you'll be setting property to the JSON null value. Using the Unknown value (?) as a constant in one of the `Set( )` overloadings causes a compilation error, because the compiler is unable to resolve the overloading to use. On successful completion, the method returns TRUE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
SetNull( INPUT property-name AS CHARACTER )
```

property-name

A CHARACTER expression naming the existing property to be set to null.

property-name is case-sensitive.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist

SetNumber( ) method (JsonArray)

Sets an element of the JsonArray to the specified value. This method is useful in situations where none of the ABL numeric data types can be used to hold a JSON numeric value. On successful execution, this method returns TRUE.

Return type: LOGICAL

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
SetNumber( INPUT index AS INTEGER,
           INPUT value AS CHARACTER )
```

index

An INTEGER that identifies the element in the array to change. Indexing into JsonArrays is 1-based.

A JsonError is raised if *index* is less than 1, is greater than the length of the JsonArray, or is the Unknown value (?).

value

A CHARACTER expression representing a numeric value to which the new property is to be set. The value should match the pattern for valid JSON number values. These may be integers ([-]dddd), decimals ([-]ddd.ddd), or scientific notation ([-]ddd[.ddd]e[+|-]ddd). If *value* is the Unknown value (?), the element is set to the JSON null value. If the value is not a valid JSON number value or is an empty string (""), a JsonError is raised.

SetNumber( ) method (JsonObject)

Sets the named property of JsonObject to the given value. This method is useful in situations where none of the ABL numeric data types can be used to hold a JSON numeric value. For example, 10E+100. On successful execution, this method returns TRUE.

Return type: LOGICAL

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
SetNumber( INPUT property-name AS CHARACTER,
           INPUT value AS CHARACTER )
```

property-name

A CHARACTER expression naming the existing property whose value is to be changed.

property-name is case-sensitive.

value

A CHARACTER expression representing a numeric value to which the new property is to be set. The value should match the pattern for valid JSON number values. These may be integers ([-]dddd), decimals ([-]ddd.ddd), or scientific notation ([-]ddd[.ddd]e[+|-]ddd). If *value* is the Unknown value (?), the element is set to the JSON null value.

A JsonError is raised if:

- *property-name* is the empty string (""), or is the Unknown value (?)
- The property does not exist
- The property value is not a valid JSON number

SetParameter( ) method (Class)

Populates elements in a `Progress.Lang.ParameterList` object.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Lang.ParameterList class

Syntax

This method supports two overloaded versions. The first version is used to initialize and populate elements in the ParameterList object. It's calling sequence is similar to the Call object's SET-PARAMETER method.

```
SetParameter ( INPUT position AS INTEGER ,
               INPUT data-type AS CHARACTER ,
               INPUT iomode AS CHARACTER ,
               INPUT parameter-value )
```

The second overloaded version may ONLY be called if SetParameter has been called before on this parameter, so that *data-type* and *iomode* are already initialized.

```
SetParameter ( INPUT position AS INTEGER ,
               INPUT parameter-value )
```

position

Identifies the one-based position that the parameter occupies in the routine signature. The AVM returns a run-time error if *position* is greater than the number of parameters defined for a `Progress.Lang.ParameterList` object, which was set either during instantiation (passed as input to the constructor) or by setting the NumParameters attribute.

data-type

A CHARACTER expression indicating the target parameter type. This is generally the data type that the invoked method expects, but can also be an AS data type to match an extended .NET data type, such as `"SHORT"`, `"BYTE"`, `"SINGLE-CHARACTER"`, and so on.

If the parameter of the method to be called is type DECIMAL, then *data-type* should evaluate to `"DECIMAL"`, even if *parameter-value* is an INTEGER or INT64. You can pass a type that matches the target using the same widening and hierarchy rules that are used during compilation. Using the exact target type will generally result in better performance.

Valid *data-type* strings are: `"CHARACTER"`, `"DATASET"`, `"DATASET-HANDLE"`, `"DATE"`, `"DATETIME"`, `"DATETIME-TZ"`, `"DECIMAL"`, `"HANDLE"`, `"INT64"`, `"INTEGER"`, `"LOGICAL"`, `"LONGCHAR"`, `"MEMPTR"`, `"RAW"`, `"ROWID"`, `"TABLE"`, `"TABLE-HANDLE"` or `"CLASS object-type-name"`. A TABLE-HANDLE can be a handle to a static or dynamic TEMP-TABLE. A DATASET-HANDLE can be a handle to a static or dynamic ProDataSet object. The BUFFER type is not supported; use a HANDLE parameter instead.

For any parameter array value, the EXTENT keyword is required when specifying *data-type*. For example, `"CHARACTER EXTENT"`, `"INTEGER EXTENT"`, `"LONGCHAR EXTENT"`, `"MEMPTR EXTENT"`, `"CLASS object-type-name EXTENT"` and so on.

For each parameter, the data type specified by the caller and the callee must be compatible with the ABL object-oriented parameter rules.

iomode

A CHARACTER expression that indicates the parameter mode and evaluates to one of the following: "INPUT", "OUTPUT", "INPUT-OUTPUT", "OUTPUT APPEND", "OUTPUT BIND", "INPUT BY-REFERENCE", "OUTPUT BY-REFERENCE", "INPUT-OUTPUT BY-REFERENCE", and "INPUT BIND". For each parameter, the mode specified by the caller and the callee must match. The AVM generates a run-time error if *iomode* is invalid.

Unlike the call object's SET-PARAMETER( ) method, with the exception of "INPUT-OUTPUT", all *iomode* combinations are separated by a blank, for example "INPUT BY-REFERENCE".

For more information on parameter modes, see the DEFINE PARAMETER statement or Parameter passing syntax reference entries.

parameter-value

A value of any data type, including the special TABLE and DATASET types. This value must be compatible with *data-type*. If *iomode* is OUTPUT or INPUT-OUTPUT, you can only specify a simple variable or property, not an expression. The variable or property must be in scope when the method is invoked using this ParameterList.

Example

The following example demonstrates using the SetParameter( ) method to construct a parameter list. The first parameter is an input parameter containing the customer number to retrieve. The second and third parameters are output parameters containing the customer name and credit limit for the retrieved customer. GetRecData is a method of class RecData which is invoked by this procedure.

```
DEFINE VARIABLE plClass AS Progress.Lang.Class.
DEFINE VARIABLE hObj AS Progress.Lang.Object.
DEFINE VARIABLE parms AS CLASS Progress.Lang.ParameterList.
DEFINE VARIABLE custname AS CHARACTER.
DEFINE VARIABLE climit AS INTEGER.
DEFINE VARIABLE custnum AS INTEGER INITIAL 8.
DEFINE VARIABLE retval AS LOGICAL.

/* Create an instance of RecData dynamically */
plClass = Progress.Lang.Class:GetClass("RecData").
hObj = plClass:NEW().

/* Call the GetRecData method dynamically */
parms = NEW Progress.Lang.ParameterList(3).
parms:SetParameter(1, "INTEGER", "INPUT", custnum).
parms:SetParameter(2, "CHARACTER", "OUTPUT", custname).
parms:SetParameter(3, "INTEGER", "OUTPUT", climit).
retval = plClass:INVOKE(hObj, "GetRecData", parms).

IF retval THEN
    MESSAGE "Customer name: " custname
            " Credit limit: " climit
            VIEW-AS ALERT-BOX.
ELSE
    MESSAGE "Customer not found"
            VIEW-AS ALERT-BOX.
```

RecData.cls:

```
USING Progress.Lang.*.
CLASS RecData INHERITS Progress.Lang.Object:

    METHOD PUBLIC LOGICAL GetRecData
        (INPUT custnum AS INTEGER,
         OUTPUT custname AS CHARACTER,
         OUTPUT creditlimit AS INTEGER):

        DEFINE VAR retval AS LOGICAL INIT TRUE.
        FIND customer WHERE (Customer.custnum = custnum) NO-ERROR.
        IF AVAILABLE Customer THEN
            ASSIGN
                custname = Customer.Name
                creditlimit = Customer.CreditLimit.
        ELSE
            retval = FALSE.

        RETURN retval.
    END METHOD.
END CLASS.
```

After running the procedure, the following output is displayed:

```
Customer name: Game Set Match  Credit limit: 15000
```

See also

Clear( ) method (Class), Invoke( ) method (Class), New( ) method, NumParameters property (ParameterList)

SetPropertyValue( ) method

The SetPropertyValue( ) method enables you to assign a value to a property even if you do not know the property's name or type at compile time.

Return type: VOID

Access: PUBLIC

Applies to: Progress.Lang.Class class

Syntax

Four overloaded versions are supported. The first version operates on a scalar instance property, where a scalar property is any data type that is not, itself, an array.

```
class-object-reference:SetPropertyValue (
  INPUT object-reference AS Progress.Lang.Object ,
  INPUT property-name AS CHARACTER, INPUT new-value AS any-data-type )
```

The second version operates on a scalar static property.

```
class-object-reference:SetPropertyValue (
  INPUT property-name AS CHARACTER, INPUT new-value AS any-data-type )
```

The third version operates on an instance array or .NET indexed property.

```
class-object-reference:SetPropertyValue (
  INPUT object-reference AS Progress.Lang.Object,
  INPUT property-name AS CHARACTER, INPUT index AS any-data-type,
  INPUT new-value AS any-data-type )
```

The fourth version operates on a static array or .NET indexed property.

```
class-object-reference:SetPropertyValue (
  INPUT property-name AS CHARACTER, INPUT index AS any-data-type,
  INPUT new-value AS any-data-type )
```

class-object-reference

Specifies a reference to an instance of `Progress.Lang.Class` for a type that exposes the property. For instance properties, `object-reference` and `class-object-reference` must be the same type, or one must inherit from the other. Both must either define the property or inherit it.

object-reference

Specifies a reference to an ABL or .NET class instance that exposes the specified property as an instance member. The compiler allows `object-reference` to be declared as any object type. At run time, the object type must resolve to the type that exposes the property. `GetPropertyValue( )` and `SetPropertyValue( )` work correctly even if the *object-reference* is not known. Alternatively, use the overloaded methods for static property.

Note: The overloads of `SetPropertyValue( )` that take an object reference can be used with a static property by setting `object-reference` to the Unknown value (?). Alternatively, use the overloads that are designed specifically for static properties.

Note: A run-time error arises if object-reference does not inherit from class-object-reference, or vice-versa.

---

property-name

Specifies a CHARACTER name of the property. The AVM evaluates *property-name* at run time.

new-value

Specifies the value that is assigned to the property. The value is converted at run time to the property's data type using the appropriate data type casting.

index

Specifies an integer expression for the index of the specified element of an ABL array property. If the property is a .NET indexed property, index is an expression for the specified element. The index for an indexed property does not have to be an integer.

any-data-type

Specifies the data type of the associated element. The set of possible data types depends on the element. For example, the index of an ABL array element can only be an integer type. However, the setting of a property, or array property element, value can be any ABL primitive data or object type.

See also

DYNAMIC-PROPERTY statement, GetPropertyValue( ) method

SetterAccessMode property

Returns an instance of `Progress.Reflect.AccessMode` that indicates the access mode for the SET accessor of the property. If the property has a SET accessor without an implementation, `SetterAccessMode` returns the access mode of the property itself. If the property has no defined SET accessor, `SetterAccessMode` returns the Unknown value (?).

Data type: Progress.Reflect.AccessMode enumeration

Access: PUBLIC Read-only

Applies to: Progress.Reflect.Property class

Severity property

A convenience property intended for ABL programmers who want to rank their application error objects according to a severity scheme. Providing this property through inheritance in the base `Progress.Lang.AppError` class allows severity ranking without having to create this property in a subclass.

Data type: INTEGER

Access: PUBLIC Readable/Writeable

Applies to: Progress.Lang.ProError class, System.Exception class (OpenEdge-enhanced)

`Progress.Lang.SysError` objects and `Progress.Lang.AppError` objects automatically created by the AVM as a result of a `RETURN ERROR` statement have no notion of error severity. For these objects, the property is set to zero (0).

.NET `Exception` objects also do not use this property.

SkipBytes( ) method (FileInputStream)

Skips and discards a specified number of bytes in the current file input stream. The method returns the number of bytes skipped. It returns 0 if the end of the stream has already been reached.

Return type: INT64

Access: PUBLIC

Applies to: Progress.IO.FileInputStream class

Syntax

```
SkipBytes( INPUT n AS INT64 )
```

n

The number of bytes to be skipped.

SkipBytes( ) method (InputStream)

Skips and discards a specified number of bytes in the current input stream. The method returns the number of bytes skipped. It returns 0 if the end of the stream has already been reached.

Return type: INT64

Access: PUBLIC

Applies to: Progress.IO.InputStream class

Syntax

```
SkipBytes( INPUT n AS INT64 )
```

n

The number of bytes to be skipped.

Notes

- This method can be overridden by the derived input stream class to handle the skipping of bytes directly from the derived input stream without calling a `Read` method.

SkipBytes( ) method (MemoryInputStream)

Skip over *n* bytes in the current memory input stream. Returns the number of bytes skipped. Returns 0 or less than *n*, if the position is at the end of the stream.

Return type: INT64

Access: PUBLIC

Applies to: Progress.IO.MemoryInputStream class

Syntax

```
SkipBytes(INPUT n AS INT64)
```

n

The number of bytes to skip over.

SoapFault property

Identifies the SOAP-FAULT object handle that contains a SOAP fault message detail.

Data type: HANDLE

Access: PUBLIC Read-only

Applies to: Progress.Lang.SoapFaultError class

If the ABL application invokes a Web service operation that returns a SOAP fault message, the AVM creates a SOAP-FAULT object. Use the SOAP-FAULT-DETAIL attribute of the SOAP-FAULT object handle to access the SOAP fault message detail.

See also

SOAP-fault object handle

Sorted property
(Windows only; GUI for .NET only)

Use this property to indicate whether you have successfully reopened the query to affect the sort based on the argument values specified in the SortRequestEventArgs object. Set to TRUE if the query was successfully reopened. Set to FALSE if the query was not reopened. The default value is TRUE.

Data type: LOGICAL

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.SortRequestEventArgs class

See also

SortRequest event

StatBaseId property

The `StatBaseId` property identifies the `StatBase` VST. This VST contains basic table and index statistics. You can pass the `StatBaseId` property to the `GetVSTHandle()` method to return the handle to the `StatBase` VST. This is a read-only property.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

String property

Returned by the GetType method on a JsonObject or JsonArray indicating that the JSON data type for the specified property or element is string.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Json.ObjectModel.JsonDataType class

Subscribe( ) method

Subscribes an event handler method or procedure to an ABL or .NET class event. This is a built-in event method for managing class events. For more information, see the Subscribe( ) event method reference entry.

Success property

An authentication status code that indicates that user credentials have been successfully authenticated.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

Subtract( ) method
(Windows only; GUI for .NET only)

Subtracts the underlying value of the first enumeration type from the second enumeration type and returns the difference as a new instance.

Return type: System.Enum class (from the .NET Framework)

Access: PUBLIC STATIC

Applies to: Progress.Util.EnumHelper class

Syntax

```
EnumHelper:Subtract ( INPUT enum1 AS CLASS System.Enum,
                      INPUT enum2 AS CLASS System.Enum )
```

enum1

Static reference to an enumeration value, or a reference to an enumeration type instance.

enum2

Static reference to an enumeration value, or a reference to an enumeration type instance.

Specify a static reference to an enumeration value using the following syntax:

```
enum-type:enum-member
```

Where *enum-type* is the type name of the enumeration and *enum-member* is a member name of the enumeration. For example:

```
System.Windows.Forms.AnchorStyles:Bottom
```

If you want to use the returned System.Enum instance as a different enumeration type, you must cast the instance to that specific type using the CAST function.

For information on the System.Enum class and .NET enumeration types, refer to the .NET Framework class library.

See also

Add( ) method (Progress.Util.EnumHelper), CAST function

SuperClass property

The object reference for the super class type information, if the user-defined class is a subclass or is an interface that the subclass implements, including an inherited interface. Otherwise, the value of this property is the Unknown value (?). If the `Progress.Lang.Class` is for an ABL class that inherits a .NET class, this property also returns the Unknown value (?).

Data type: Progress.Lang.Class class

Access: PUBLIC Read-only

Applies to: Progress.Lang.Class class

SyncToSelectedRow property
(Windows only; GUI for .NET only)

Indicates if the query position (cursor) should be repositioned to the selected row after fetching data from another row. The default value is TRUE. Set to FALSE to avoid repositioning the bound query cursor to the selected row after fetching data from another row.

Data type: LOGICAL

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

After the BindingSource fetches a new row, the query cursor is set back to the selected position. This selected position matches the Position property (BindingSource) and coincides with the user's selected row. Normally this repositioning is necessary in order to guarantee that when an event happens, the selected row is in the record buffer when the event handler code runs. For example, when the RowValidating event occurs, it is desirable to have the modified row in the buffer when the RowValidating event handler runs. Without repositioning, the behavior is not consistent.

However, there are some operations that do not require this repositioning. One application is when using the grouping operation of a Telerik RadGridView. To avoid the repositioning, set SyncToSelectedRow to FALSE when the grid fires the GroupByChanging event and reset it to TRUE when the GroupByChanged event fires.

Another use case for setting SyncToSelectedRow to FALSE is when report-mode is set to TRUE on a BindingSource constructor. Report-mode is used when generating reports in batch mode using a non-UI bound reporting object.

TableName property

The name of the database table that has the specified lock conflict.

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Lang.LockConflict class

See also

CallStack property, Device property, User property

TableSchema property
(Windows only; GUI for .NET only)

References a Progress.Data.TableDesc class instance that specifies schema information for the data associated with a BindingSource object.

Note: This property is designed for use by the Visual Designer. Though you can do so, it is not expected that you will use this in your own code.

Data type: Progress.Data.TableDesc class

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

In the Visual Designer, when you define the logical schema (tables and fields) for a BindingSource object, the Visual Designer's auto-generated code stores this information using the TableSchema property. At run time, the actual data source object (such as an ABL query or ProDataSet) is specified by setting the Handle property of the BindingSource. The generated code then matches up the logical schema that you defined with the data source object in order to display the correct data. For more information, see the reference entry for the Progress.Data.TableDesc class in this book.

TableStatId property

The `TableStatId` property identifies the `TableStat` VST. This VST displays statistics on the number of accesses to a specific range of tables. You can pass the `TableStatId` property to the `GetVSTHandle()` method to return the handle to the `TableStat` VST. This is a read-only property.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

Tag property (Progress.Data.BindingSource)
(Windows only; GUI for .NET only)

Allows you to assign and return an arbitrary .NET object associated with a `BindingSource`.

Data type: `System.Object` class (from the .NET Framework)

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

One use of this property is to store a character string containing some descriptive information about the `BindingSource` instance. For example, you might assign a character expression directly to this property, which ABL boxes as a `System.String` before assigning it to the property.

TempTableCount property

Returns the number of instances of static and dynamic temp-tables in the current session, You will have two instances of the same temp-table when there are two instances of a persistent procedure containing a static temp-table.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.TempTableInfo class

TempTablePeak property

Returns the highest temp-table ID used in a temp-table session irrespective of whether the temp-table is created statically or dynamically. This might help when you make adjustments to the `-tttablerangesize` parameter.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.TempTableInfo class

ThreadId property
(PAS for OE only)

Identifies the Progress Application Server thread associated with a given request or response. This property holds the value used by the multi-session agent, not the session manager. If there is no valid response or request running in the session, the property defaults to the Unknown value (?).

Data type: INTEGER

Access: PUBLIC Read-only

Applies to: Progress.Lang.OERequestInfo class

ToggleFlag( ) method

Returns a flag enum instance with the state of the indicated flag(s) toggled. In other words, the flag is unset if it was previously set, and set if it was previously unset.

Return type: Flag enum type

Access: PUBLIC

Applies to: Progress.Reflect.Flags enumeration, Progress.Reflect.ParameterMode enumeration, any user-defined flag enum type

Syntax

```
ToggleFlag ( INPUT flag AS flag-enum-type )
```

flag

A reference to a flag enum instance. The enum type (*flag-enum-type*) of this instance must match the type of the instance the method is invoked on.

The following example results in `vReflectFlag`, an instance of `Progress.Reflect.Flags`, with the `Public` flag set and the `Protected` flag unset:

```
DEFINE VARIABLE vReflectFlag AS Flags.
vReflectFlag = Flags:Protected.

/* These two lines toggle the status of the Public and
   Protected flags without affecting the status of any of
   the other flags in vReflectFlag. */
vReflectFlag = vReflectFlag:ToggleFlag(Flags:Public).
vReflectFlag = vReflectFlag:ToggleFlag(Flags:Protected).
```

You can use the bitwise XOR operator to achieve the same result. For example, replacing the last line with `vReflectFlag = vReflectFlag XOR Flags:Protected` also toggles the status of the `Protected` flag.

Notes

- This method is available for all built-in flag enums, and the compiler automatically generates it for all user-defined flag enums. For example, this code excerpt uses a user-defined `Permissions` flag enum, initializes `vPerm` with the `Read` flag set, and then uses `ToggleFlag( )` to set the `Create` flag, resulting in an enum instance with both the `Read` and `Create` flags set:

```
ENUM Permission FLAGS:
    DEFINE ENUM None = 0
                Read
                Write
                ReadWrite = Read,Write
                Create
                Delete.
END ENUM.
```

```
DEFINE VARIABLE vPerm AS Permission.
vPerm = Permission:Read.

/* This toggles the Create flag without affecting the status of
   any of the other flags in vPerm. */
vPerm = vPerm:ToggleFlag(Permission:Create).
```

See also

SetFlag( ) method, UnsetFlag( ) method, XOR operator (bitwise)

ToObject( ) method

Returns an enum instance given the enum type and enum member name or numeric value.

Return type: Progress.Lang.Enum

Access: PUBLIC STATIC FINAL

Applies to: Progress.Lang.Enum class

Syntax

```
ToObject ( INPUT enum-type-name AS CHARACTER,
           INPUT enum-member-name AS CHARACTER )

ToObject ( INPUT enum-type-name AS CHARACTER,
           INPUT value AS INT64 )
```

enum-type-name

The name of an enum type.

enum-member-name

The name of an enum member of the specified enum type. For a flag enum type, this can be a comma-delimited list of enum members.

value

The underlying numeric value of an enum member of the specified enum type. For a flag enum type, this value can represent two or more enum members.

The method returns an error if a member name or numeric value does not correspond to a member of the enum type. The exception to this is that you can pass a value of 0 for a flag enum type even if no member has been explicitly defined for 0.

This code excerpt creates an instance of the user-defined `Permission` flag enum and assigns `vPerm` the members `Write` and `Create`. Both versions of the final statement have the same result:

```
ENUM Permission FLAGS:
    DEFINE ENUM None = 0
                Read = 0x01
                Write = 0x02
                ReadWrite = 0x03
                Create = 0x04
                Delete = 0x08.
END ENUM.
```

```
USING Progress.Lang.*.

DEFINE VARIABLE vPerm AS Permission.

vPerm = CAST(Enum:ToObject(Permission, "Write,Create"), Permission).
vPerm = CAST(Enum:ToObject(Permission, 0x06), Permission).
```

Notes

- Because `ToObject( )` returns a weakly typed instance, you must use CAST, as shown in the example above, if you want to assign the method's return value to a variable defined as a specific enum type. To create a strongly typed enum instance, use the `GetEnum( )` method.

See also

GetEnum( ) method

ToString( ) method (Object)

Returns information from the specified class or enum instance (both ABL and .NET), depending on the object type.

Return type: CHARACTER

Access: PUBLIC

Applies to: Progress.Lang.Object class, System.Object class (from the .NET Framework)

Syntax

```
ToString( )
```

This method is also defined on the .NET root class, System.Object, which overrides the method inherited in the ABL class structure from Progress.Lang.Object.

On an ABL class, or on an ABL-derived .NET class instance that does not override ToString( ), this method returns the fully-qualified class or interface type name of the object followed by a unique object identifier, separated by an underscore. You typically override this method in a user-defined class to provide additional contextual information.

On an ABL enum instance, this method returns the name of the enumeration member. For example, if applied to `Progress.Reflect.Flags:Public`, the method returns "Public". In the case of a flag enum instance with multiple flags set, the method returns a comma-delimited list of all the flags that are set. If there are no flags set or the underlying numeric value is 0 and there is no flag defined for 0, the method returns the empty string ("").

On a pure .NET class, the information returned depends on the class type. Also, for any System.Object that boxes a .NET mapped data type (such as a System.Int32), the method returns a character string that represents the value of the mapped data type instance. (For more information on .NET mapped data types, see the Data types reference entry.) For all other .NET types, the information returned depends on how the .NET type overrides the ToString( ) method. For example, on a .NET generic type instance, the method returns a string with the following syntax:

```
namespace.object-name‘integer[type-name [, type-name ] ...]
```

Where, *namespace* is the .NET namespace, *object-name* is the .NET generic class or interface name up to the left angle bracket (<) of the generic type name, *integer* after the backquote (‘) is the number of type parameters in the generic type, and *type-name* is the .NET type name substituted for each type parameter in the constructed type. For more information on .NET constructed type names for generic types, see the Data types reference entry.

So, invoking the method on an instance of the following .NET generic type:

```
"System.Collection.Generic.List<CHARACTER>"
```

Returns the following string:

```
System.Collections.Generic.List`1[System.String]
```

Note: This method is used by some ABL string functions and statements, such as the QUOTER and STRING functions and the MESSAGE and PUT statements.

ToString( ) method (Reflect)

Returns a string describing the constructor, event, method, parameter, property, or data member described by the object the method is invoked on.

Return type: CHARACTER

Access: PUBLIC

Applies to: Progress.Reflect.Constructor class, Progress.Reflect.Event class, Progress.Reflect.Method class, Progress.Reflect.Parameter class, Progress.Reflect.Property class, Progress.Reflect.Variable class

Syntax

```
ToString( )
```

The format of the string is similar to the definition statement of the particular class member. Following are examples for each possible object type :

- **Constructor**: "CONSTRUCTOR PUBLIC Customer (INPUT name AS CHARACTER)"
- **Event**: "PUBLIC EVENT NewCustomer VOID (INPUT name AS CHARACTER)"
- **Method**: "METHOD PUBLIC INTEGER CreateCustomer(INPUT name AS CHARACTER)"
- **Parameter**: "OUTPUT names AS CHARACTER EXTENT 10"
- **Property**: "PROTECTED PROPERTY dd AS DATE GET"
- **Variable**: "PUBLIC STATIC VARIABLE cc AS CHARACTER"

Notes

- The property example given above shows a read-only property. The bodies of a property's `GET` and `SET` methods are never shown.

TransId property

The `TransId` property identifies the `TransId` VST. This VST displays information about the transaction such as transaction number, state, start time, duration, user number, coordinator name, and transaction. You can pass the `TransId` property to the `GetVSTHandle()` method to return the handle to `Trans` VST. This is a read-only property.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

TypeName property

The type name of the class or interface, which consists of the package and the class or interface name.

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Lang.Class class

See also

Type-name syntax

UnknownUser property

An authentication status code that indicates that the user account does not exist in the user account system defined in the OpenEdge domain.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *ABL Reference*)

UnsetFlag( ) method

Returns a flag enum instance with the indicated flag(s) unset.

Return type: Flag enum type

Access: PUBLIC

Applies to: Progress.Reflect.Flags enumeration, Progress.Reflect.ParameterMode enumeration, any user-defined flag enum type

Syntax

```
UnsetFlag ( INPUT flag AS flag-enum-type )
```

flag

A reference to a flag enum instance. The enum type (*flag-enum-type*) of this instance must match the type of the instance the method is invoked on.

The following example results in `vReflectFlag`, an instance of `Progress.Reflect.Flags`, with only the `Public` flag set:

```
DEFINE VARIABLE vReflectFlag AS Flags.
vReflectFlag = Flags:Public.
vReflectFlag = vReflectFlag:SetFlag(Flags:Protected).

/* This unsets the Protected flag without affecting the status of
   any of the other flags in vReflectFlag. */
vReflectFlag = vReflectFlag:UnsetFlag(Permission:Protected).
```

You can use a bitwise NOT operation to achieve the same result. For example, replacing the last line with `vReflectFlag = vReflectFlag AND NOT Flags:Protected` also unsets the `Protected` flag.

Notes

- This method is available for all built-in flag enums, and the compiler automatically generates it for all user-defined flag enums. For example, this code excerpt uses a user-defined `Permission` flag enum, initializes `vPerm` with the `Create` and `Delete` flags set, and then uses `UnsetFlag( )` to unset the `Delete` flag:

```
ENUM Permission FLAGS:
    DEFINE ENUM None = 0
                Read
                Write
                ReadWrite = Read,Write
                Create
                Delete.
END ENUM.
```

```
DEFINE VARIABLE vPerm AS Permission.

vPerm = Permission:Create.
vPerm = vPerm:SetFlag(Permission:Delete).

/* This unsets the Delete flag without affecting the status of
   any of the other flags in vPerm. */
vPerm = UnsetFlag(Permission:Delete).
```

See also

SetFlag( ) method, NOT operator (bitwise), ToggleFlag( ) method

---

Unsubscribe( ) method

Removes a method or procedure as a handler for an ABL or .NET class event. This is a built-in event method for managing class events. For more information, see the Unsubscribe( ) event method reference entry.

UseFullFieldNames property
(Windows only; GUI for .NET only)

`UseFullFieldNames` allows you to display duplicate fields when there are duplicate field names from different tables in a joined query. `UseFullFieldNames` is only relevant when the `ProBindingSource` object is bound to a joined query. The default value is FALSE.

Data type: LOGICAL

Access: PUBLIC Readable/Writeable

Applies to: Progress.Data.BindingSource class

If setting `UseFullFieldNames` to TRUE, you reference all columns in the `ProBindingSource` using the format:

```
buffer-name:field-name
```

The following example code illustrates how to use this format:

```
oArray[1] = NEW Progress.Data.ColumnPropDesc("Customer:CustNum","Number",
  Progress.Data.DataType:INTEGER).

myTextBox:DataBindings:Add("Text",pbs,"Bin:BinNum").
```

Note: When using the `ProBindingSource:SetFields()` method, you specify the field name using the ABL format, not the above format. For example:

```
myBS:SetFields("Customer.CustNum,Customer.Name,Order.CustNum,Order.OrderNum","","").
```

User property

The name of the user that is currently holding the lock that is causing the specified lock conflict.

Data type: CHARACTER

Access: PUBLIC Read-only

Applies to: Progress.Lang.LockConflict class

See also

CallStack property, Device property, TableName property

UserIndexStatId property

The `UserIndexStatId` property identifies the `UserIndexStat` VST. This VST displays statistics on the number of accesses to a specific range of indexes. You can pass the `UserIndexStatId` property to the `GetVSTHandle()` method to return the handle to the `UserIndexStat` VST. This is a read-only property.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

UserIOId property

The `UserIOId` property identifies the `UserIO` VST. This VST contains information about the temp-table database input/output operations, such as user number and name; and the number of accesses, reads, and writes. You can pass the `UserIOId` property to the `GetVSTHandle()` method to return the handle to `UserIO` VST. This is a read-only property.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

UserTableStatId property

The `UserTableStatId` property identifies the `UserTableStat` VST. This VST displays statistics on the number of accesses to a specific range of tables. You can pass the `UserTableStatId` property to the `GetVSTHandle()` method to return the handle to `UserTableStat` VST. This is a read-only property.

Data type: INTEGER

Access: PUBLIC STATIC

Applies to: Progress.Database.VSTTableId class

VersionInfo property

An object reference to a `Progreses.Lang.OEVersionInfo` class that identifies the OpenEdge version of a given OpenEdge session.

Data type: Progress.Lang.OEVersionInfo class

Access: PUBLIC Read-only

Applies to: Progress.Lang.OERequestInfo class

You can access this property in an:

- **Progress Application Server session** — Using the CURRENT-REQUEST-INFO and CURRENT-RESPONSE-INFO attributes on the SESSION system handle
- **ABL client session** — Using the REQUEST-INFO and RESPONSE-INFO attributes on the server object handle used to invoke the remote request, and on the asynchronous request object handle returned for an asynchronous request

The *REQUEST-INFO attributes provide version information for the associated Progress Application Server client; the *RESPONSE-INFO attributes provide version information for the associated Progress Application Server.

See also

CURRENT-REQUEST-INFO attribute, CURRENT-RESPONSE-INFO attribute, REQUEST-INFO attribute, RESPONSE-INFO attribute

Write( ) method (MemoryOutputStream)

Writes data from specified input parameters to a MemoryOutputStream object. Returns the total number of characters written for CHARACTER and LONGCHAR data parameter, or the total number of bytes written for a MEMPTR parameter.

Return type: INT64

Access: PUBLIC

Applies to: Progress.IO.MemoryOutputStream class

Syntax

```
Write(INPUT source-data AS MEMPTR, INPUT offset AS INT64,INPUT length AS INT64)
Write(INPUT source-data AS  CHARACTER)
Write(INPUT source-data AS  CHARACTER, INPUT incldNull AS LOGICAL)
Write(INPUT source-data AS LONGCHAR)
Write(INPUT source-data AS LONGCHAR, INPUT incldNull AS LOGICAL))
```

source-data

A CHARACTER, LONGCHAR, or MEMPTR variable containing the data the method will write to the MemoryOutputStream object.

Note: The AVM converts LONGCHAR data values to the code page specified by the `-cpinternal` parameter.

offset

Indicates which byte of the source-data MEMPTR the method should start with when writing to the current MemoryOutputStream object.

length

Number of bytes to write to the MemoryOutputStream object.

incldNull

If set to TRUE, includes the null terminator in CHARACTER or LONGCHAR data written to the current MemoryOutputStream object.

Example

```
DEFINE VAR outMs  AS Progress.io.MemoryOutputStream.
DEFINE VAR mem    AS MEMPTR.
DEFINE VAR len    AS INT64.
DEFINE VAR ret    AS INT64.
DEFINE VAR mylc   AS LONGCHAR.

outMs  = NEW Progress.io.MemoryOutputStream().
mylc = "123456789".
ret = outMs:Write(mylc,yes).
MESSAGE ret " == " outMs:BytesWritten " == " 10 VIEW-AS ALERT-BOX.

outMs:Clear().

outMs  = NEW Progress.io.MemoryOutputStream().
mylc = "123456789".
ret = outMs:Write(mylc). //null terminator will be exclued
MESSAGE ret " == " outMs:BytesWritten " == " 9 VIEW-AS ALERT-BOX.

outMs:Clear().

SET-SIZE(mem) = 11.
PUT-STRING(mem,1) = "123456789".
ret = outMs:Write(mem, 1, LENGTH(STRING(mem))).
MESSAGE "123456789 == " STRING(outMs:Data)  VIEW-AS ALERT-BOX.
```

Note

The following table illustrates values that result from constructors and `Write( )`/`Clear( )` methods:

Method/Constructor	Values	
	Obj:Length	Obj:BytesWritten
`Obj = NEW MemoryOutputStream()`	0	0
`Obj:Write(“12345”)`	5	5
`Obj:Clear()`	0	0
`Obj = NEW MemoryOutputStream(100)`	100	0
`Obj:Write(“12345678”)`	100	8
`Obj:Clear()`	100	0

Write( ) method (JsonArray)

Serializes the `JsonArray` and any `JsonConstructs` it references. This is a recursive process resulting in a JSON representation of the entire set of `JsonObjects` and `JsonArrays`. On successful completion this method returns TRUE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
Write( INPUT stream-handle AS HANDLE)

Write( INPUT stream-handle AS HANDLE,
       INPUT formatted AS LOGICAL)

Write( INPUT stream-handle AS HANDLE,
       INPUT formatted AS LOGICAL,
       INPUT encoding AS CHARACTER)

Write( INPUT-OUTPUT longchar AS LONGCHAR)

Write( INPUT-OUTPUT longchar AS LONGCHAR,
       INPUT formatted AS LOGICAL)

Write( INPUT-OUTPUT longchar AS LONGCHAR,
       INPUT formatted AS LOGICAL,
       INPUT encoding AS CHARACTER)

Write( INPUT memptr AS MEMPTR)

Write( INPUT memptr AS MEMPTR,
       INPUT formatted AS LOGICAL)

Write( INPUT memptr AS MEMPTR,
       INPUT formatted AS LOGICAL,
       INPUT encoding AS CHARACTER)

Write( OUTPUT character AS CHARACTER)

Write( OUTPUT character AS CHARACTER,
       INPUT formatted AS LOGICAL)
```

stream-handle

HANDLE variable that specifies a stream object handle. If the parameter is another handle type, a JsonError error is raised.

longchar

A LONGCHAR variable to contain the JSON string in memory.

The AVM saves the JSON string to the LONGCHAR variable in the code page that corresponds to the character encoding you specify in the encoding option. If you do not specify a character encoding for the JSON string, the AVM saves the LONGCHAR variable as "UTF-8".

If the LONGCHAR variable's code page is fixed using the FIX-CODEPAGE statement and the fixed code page is not equivalent to the character encoding, this method raises a JsonError error.

memptr

A MEMPTR variable to contain the JSON string in memory. If you do not specify the encoding parameter, the AVM encodes the text written to the MEMPTR as "UTF-8". This method allocates the required amount of memory for the JSON string and sets the size of the variable. If the MEMPTR contains allocated memory when the method is called, that memory may be reallocated during the execution of the method. When you are finished using the MEMPTR, you must free the associated memory, by executing SET-SIZE(*memptr*) = 0 on the MEMPTR.

character

A CHARACTER variable to contain the JSON string in memory. An error is raised if the JSON string cannot be transcoded from Unicode to `-cpinternal`.

formatted

A LOGICAL expression where TRUE directs the AVM to format the JSON string in a hierarchical manner using extra white space, carriage returns, and line feeds. The default value is FALSE.

If you specify the Unknown value (?), the method uses the default value of FALSE.

encoding

A CHARACTER expression indicating the Unicode Transformation Format (UTF) of the JSON string to be written. Valid values for UTF are "`UTF-8`", "`UTF-16`", "`UTF-16BE`", "`UTF-16LE`", "`UTF-32`", "`UTF-32BE`", and "`UTF-32LE`". If the value is the empty string (""), the Unknown value (?), or not given, the parameter defaults to "`UTF-8`".

Notes

- Binary types, including BLOB, RAW, and ROWID are base64-encoded/decoded. See Data type mapping in *Use JSON with ABL Applications* for more information.

Write( ) method (JsonConstruct)

Serializes the JsonConstruct (JsonObject or JsonArray) and any JsonConstructs it references. This is a recursive process resulting in a JSON representation of the entire set of JsonObjects and JsonArrays. On successful execution the method returns TRUE.

Return type: LOGICAL

Access: PUBLIC ABSTRACT

Applies to: Progress.Json.ObjectModel.JsonConstruct class

Syntax

```
Write( INPUT stream-handle AS HANDLE,
       INPUT formatted AS LOGICAL)

Write( INPUT stream-handle AS HANDLE,
       INPUT formatted AS LOGICAL,
       INPUT encoding AS CHARACTER)

Write( INPUT-OUTPUT longchar AS LONGCHAR)

Write( INPUT-OUTPUT longchar AS LONGCHAR,
       INPUT formatted AS LOGICAL)

Write( INPUT-OUTPUT longchar AS LONGCHAR,
       INPUT formatted AS LOGICAL,
       INPUT encoding AS CHARACTER)

Write( INPUT memptr AS MEMPTR)

Write( INPUT memptr AS MEMPTR,
       INPUT formatted AS LOGICAL)

Write( INPUT memptr AS MEMPTR,
       INPUT formatted AS LOGICAL,
       INPUT encoding AS CHARACTER)

Write( OUTPUT character AS CHARACTER)

Write( OUTPUT character AS CHARACTER,
       INPUT formatted AS LOGICAL)
```

stream-handle

HANDLE variable that specifies a stream object handle. If the parameter is another handle type, a JsonError error is raised.

longchar

A LONGCHAR variable to contain the JSON string in memory.

The AVM saves the JSON string to the LONGCHAR variable in the code page that corresponds to the character encoding you specify in the encoding option. If you do not specify a character encoding for the JSON string, the AVM saves the LONGCHAR variable in "UTF-8".

If the LONGCHAR variable's code page is fixed using the FIX-CODEPAGE statement and the fixed code page is not equivalent to the character encoding, this method raises a JsonError error.

memptr

A MEMPTR variable to contain the JSON string in memory. If you do not specify the encoding parameter, the AVM encodes the text written to the MEMPTR as "UTF-8". This method allocates the required amount of memory for the JSON string and sets the size of the variable. If the MEMPTR contains allocated memory when the method is called, that memory may be reallocated during the execution of the method. When you are finished using the MEMPTR, you must free the associated memory, by executing SET-SIZE(memptr) = 0 on the MEMPTR.

character

A CHARACTER variable to contain the JSON string in memory. An error is raised if the JSON string cannot be transcoded from Unicode to `cpinternal`.

formatted

An optional LOGICAL expression where TRUE directs the AVM to format the JSON string in a hierarchical manner using extra white space, carriage returns, and line feeds. The default value is FALSE.

If you specify the Unknown value (?), the method uses the default value of FALSE.

encoding

An optional string indicating the Unicode Transformation Format (UTF) of the JSON string to be written. Valid values for UTF are "`UTF-8`", "`UTF-16`", "`UTF-16BE`", "`UTF-16LE`", "`UTF-32`", "`UTF-32BE`", and "`UTF-32LE`". If the value is the empty string (""), the Unknown value (?) or not given, the parameter defaults to "`UTF-8`".

Write( ) method (JsonObject)

Serializes the `JsonObect` and any `JsonConstructs` it references. This is a recursive process resulting in a JSON representation of the entire set of `JsonObjects` and `JsonArrays`. On successful execution the method returns TRUE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
Write( INPUT stream-handle AS HANDLE )

Write( INPUT stream-handle AS HANDLE,
       INPUT formatted AS LOGICAL )

Write( INPUT stream-handle AS HANDLE,
       INPUT formatted AS LOGICAL,
       INPUT encoding AS CHARACTER )

Write( INPUT-OUTPUT longchar AS LONGCHAR )

Write( INPUT-OUTPUT longchar AS LONGCHAR,
       INPUT formatted AS LOGICAL )

Write( INPUT-OUTPUT longchar AS LONGCHAR,
       INPUT formatted AS LOGICAL,
       INPUT encoding AS CHARACTER )

Write( INPUT memptr AS MEMPTR )

Write( INPUT memptr AS MEMPTR,
       INPUT formatted AS LOGICAL )

Write( INPUT memptr AS MEMPTR,
       INPUT formatted AS LOGICAL,
       INPUT encoding AS CHARACTER )

Write( OUTPUT character AS CHARACTER )

Write( OUTPUT character AS CHARACTER,
       INPUT formatted AS LOGICAL )
```

stream-handle

HANDLE variable that specifies a stream object handle. If the parameter is another handle type, a `JsonError` error is raised.

longchar

A LONGCHAR variable to contain the JSON string in memory.

The AVM saves the JSON string to the LONGCHAR variable in the code page that corresponds to the character encoding you specify in the encoding option. If you do not specify a character encoding for the JSON string, the AVM saves the LONGCHAR variable in "`UTF-8`".

If the LONGCHAR variable's code page is fixed using the `FIX-CODEPAGE` statement and the fixed code page is not equivalent to the character encoding, this method raises a `JsonError` error.

memptr

A MEMPTR variable to contain the JSON string in memory. If you do not specify the encoding parameter, the AVM encodes the text written to the MEMPTR as "`UTF-8`". This method allocates the required amount of memory for the JSON string and sets the size of the variable. If the MEMPTR contains allocated memory when the method is called, that memory may be reallocated during the execution of the method. When you are finished using the MEMPTR, you must free the associated memory, by executing `SET-SIZE(`*`memptr`*`) = 0` on the MEMPTR.

character

A CHARACTER variable to contain the JSON string in memory. An error is raised if the JSON string cannot be transcoded from Unicode to -cpinternal.

formatted

A LOGICAL expression where TRUE directs the AVM to format the JSON string in a hierarchical manner using extra white space, carriage returns, and line feeds. The default value is FALSE.

If you specify the Unknown value (?), the method uses the default value of FALSE.

encoding

A CHARACTER expression indicating the Unicode Transformation Format (UTF) of the JSON string to be written. Valid values for UTF are "UTF-8", "UTF-16", "UTF-16BE", "UTF-16LE", "UTF-32", "UTF-32BE", and "UTF-32LE". If the value is the empty string (""), the Unknown value (?), or not given, the parameter defaults to "UTF-8".

Notes

- Binary types, including BLOB, RAW, and ROWID are base64-encoded/decoded. See Data type mapping in *Use JSON with ABL Applications* for more information.

Write( ) method (OutputStream)

This method writes data from the indicated inputs to the output stream. It returns the total number of characters written for CHARACTER and LONGCHAR data inputs, or the total number of bytes written for a MEMPTR input.

Return type: INT64

Access: PUBLIC

Applies to: Progress.IO.OutputStream class

Syntax

```
ABSTRACT Write( INPUT source-data AS CHARACTER )

ABSTRACT Write( INPUT source-data AS LONGCHAR )

ABSTRACT Write( INPUT source-data AS MEMPTR,
                INPUT offset AS INT64,
                INPUT length AS INT64 )

Write( INPUT source-data AS MEMPTR )
```

length

Indicates how many bytes of the *source-data* MEMPTR the method should write to the current output stream.

offset

Indicates which byte of the *source-data* MEMPTR the method should start with when writing to the current output stream.

source-data

A CHARACTER, LONGCHAR, or MEMPTR variable containing the data the method will write to the output stream.

Notes

- All of the overloads of this method are defined as abstract except for `Write( INPUT target AS MEMPTR )`. The behavior described for the parameters of the abstract methods are provided as guidelines.
- If the size of a MEMPTR was allocated outside of the AVM (in a Windows DLL or UNIX shared library, for example), you must use the SET-SIZE statement to initialize the MEMPTR before calling the `Write( )` method.

WriteByte( ) method

Write the unsigned 1-byte value of *i* to the current MemoryOutputStream object. The AVM returns a runtime overflow error if *i* cannot fit into 1 byte or be represented as a byte.

Note: This method is equivalent to the `PUT-BYTE` statement, which applies to MEMPTR variables but not to MemoryOutputStream objects.

Return type: VOID

Access: PUBLIC

Applies to: Progress.IO.MemoryOutputStream class

Syntax

```
WriteByte(INPUT i AS INT64)
```

WriteDouble( ) method

Write the 8-byte floating-point-value of a DECIMAL *d* to the current MemoryOutputStream object.

Note: This method is equivalent to the `PUT-DOUBLE` statement, which applies to MEMPTR variables but not to MemoryOutputStream objects.

Return type: VOID

Access: PUBLIC

Applies to:

Syntax

```
WriteDouble(INPUT d AS DECIMAL)
```

WriteFile( ) method (JsonArray)

Serializes the JsonArray and any JsonConstructs it references to an output file. This is a recursive process resulting in a JSON representation of the entire set of JsonObjects and JsonArrays. On successful completion this method returns TRUE.

The method returns TRUE on successful serialization.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
WriteFile( INPUT file-name AS CHARACTER )

WriteFile( INPUT file-name AS CHARACTER,
           INPUT formatted AS LOGICAL )

WriteFile( INPUT file-name AS CHARACTER,
           INPUT formatted AS LOGICAL,
           INPUT encoding AS CHARACTER )
```

file-name

A CHARACTER expression that specifies the name of a file to which the AVM writes the JSON string. You can specify the absolute path or a path relative to the current working directory. If a file with the specified name already exists, the AVM verifies that the file is writeable and overwrites the file.

formatted

An optional LOGICAL expression where TRUE directs the AVM to format the JSON string in a hierarchical manner using extra white space, carriage returns, and line feeds. The default value is FALSE.

If you specify the Unknown value (?), the method uses the default value of FALSE.

encoding

An optional string indicating the Unicode Transformation Format of the JSON string to be written. Valid values for UTF are "UTF-8", "UTF-16", "UTF-16BE", "UTF-16LE", "UTF-32", "UTF-32BE", and "UTF-32LE". If the value is the empty string (""), the Unknown value (?), or not given, the parameter defaults to "UTF-8".

WriteFile( ) method (JsonConstruct)

Serializes the JsonConstruct (JsonObject or JsonArray) and any JsonConstructs it references to an output file. This is a recursive process resulting in a JSON representation of the entire set of JsonObjects and JsonArrays. On successful execution the method returns TRUE.

Return type: LOGICAL

Access: PUBLIC ABSTRACT

Applies to: Progress.Json.ObjectModel.JsonConstruct class

Syntax

```
WriteFile( INPUT file-name AS CHARACTER )

WriteFile( INPUT file-name AS CHARACTER,
           INPUT formatted AS LOGICAL )

WriteFile( INPUT file-name AS CHARACTER,
           INPUT formatted AS LOGICAL,
           INPUT encoding AS CHARACTER )
```

file-name

A CHARACTER expression that specifies the name of a file to which the AVM writes the JSON string. You can specify an absolute pathname or a pathname relative to the current working directory. If a file with the specified name already exists, the AVM verifies that the file is writeable and overwrites the file.

formatted

An LOGICAL expression where TRUE directs the AVM to format the JSON string in a hierarchical manner using extra white space, carriage returns, and line feeds. The default value is FALSE.

If you specify the Unknown value (?), the method uses the default value of FALSE.

encoding

A CHARACTER expression indicating the Unicode Transformation Format of the JSON string to be written. Valid values for UTF are "UTF-8", "UTF-16", "UTF-16BE", "UTF-16LE", "UTF-32", "UTF-32BE", and "UTF-32LE". If the value is the empty string (""), the Unknown value (?), or not given, the parameter defaults to "UTF-8". If the value is any other value an error is raised.

WriteFile( ) method (JsonObject)

Serializes the JsonObect and any JsonConstructs it references to an output file. This is a recursive process resulting in a JSON representation of the entire set of JsonObjects and JsonArrays. On successful execution the method returns TRUE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
WriteFile( INPUT file-name AS CHARACTER )

WriteFile( INPUT file-name AS CHARACTER,
           INPUT formatted AS LOGICAL )

WriteFile( INPUT file-name AS CHARACTER,
           INPUT formatted AS LOGICAL,
           INPUT encoding AS CHARACTER )
```

file-name

A CHARACTER expression that specifies the name of a file to which the AVM writes the JSON string. You can specify an absolute pathname or a pathname relative to the current working directory. If a file with the specified name already exists, the AVM verifies that the file is writeable and overwrites the file.

formatted

An LOGICAL expression where TRUE directs the AVM to format the JSON string in a hierarchical manner using extra white space, carriage returns, and line feeds. The default value is FALSE.

If you specify the Unknown value (?), the method uses the default value of FALSE.

encoding

A CHARACTER expression indicating the Unicode Transformation Format of the JSON string to be written. Valid values for UTF are "UTF-8", "UTF-16", "UTF-16BE", "UTF-16LE", "UTF-32", "UTF-32BE", and "UTF-32LE". If the value is the empty string (""), the Unknown value (?), or not given, the parameter defaults to "UTF-8". If the value is any other value an error is raised.

WriteFloat( ) method

Write the signed 4-byte value of *d* (as a FLOAT) to the current MemoryOutputStream object.

Note: This method is equivalent to the PUT-FLOAT statement, which applies to MEMPTR variables but not to MemoryOutputStream objects.

Return type: VOID

Access: PUBLIC

Applies to: Progress.IO.MemoryOutputStream class

Syntax

```
WriteFloat(INPUT d AS DECIMAL)
```

WriteInt64( ) method

Write the signed 8-byte value of *i* to the current MemoryOutputStream object.

Note: This method is equivalent to the PUT-INT64 statement, which applies to MEMPTR variables but not to MemoryOutputStream objects.

Return type: VOID

Access: PUBLIC

Applies to: Progress.IO.MemoryOutputStream class

Syntax

```
WriteInt64(INPUT i AS INT64)
```

WriteLong( ) method

Write the signed 4-byte value of *i* to the current MemoryOutputStream object. The AVM will return a runtime overflow error if the value *i* cannot fit into 4 bytes or be represented as a LONG type.

Note: This method is equivalent to the PUT-LONG statement, which applies to MEMPTR variables but not to MemoryOutputStream objects.

Return type: VOID

Access: PUBLIC

Applies to: Progress.IO.MemoryOutputStream class

Syntax

```
WriteLong(INPUT i AS INT64)
```

WriteShort( ) method

Write the signed 2-byte value of *i* (as a SHORT) to the current MemoryOutputStream object. The AVM will return a runtime overflow error if the value *i* cannot fit 2 into bytes or be represented as a SHORT type.

Note: This method is equivalent to the PUT-SHORT statement, which applies to MEMPTR variables but not to MemoryOutputStream objects.

Return type: INT64

Access: PUBLIC

Applies to: Progress.IO.MemoryOutputStream class

Syntax

```
WriteShort(INPUT i AS INT64)
```

WriteStream( ) method (JsonArray)

Serializes the JsonArray and any JsonConstructs it references to a statically defined output stream. This is a recursive process resulting in a JSON representation of the entire set of JsonObjects and JsonArrays. On successful completion this method returns TRUE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonArray class

Syntax

```
WriteStream(INPUT stream AS CHARACTER)

WriteStream(INPUT stream AS CHARACTER,
            INPUT formatted AS LOGICAL)

WriteStream(INPUT stream AS CHARACTER,
            INPUT formatted AS LOGICAL,
            INPUT encoding AS CHARACTER)
```

stream

A CHARACTER expression that specifies the name of a stream to which the AVM writes the JSON string. If you specify the empty string (""), the AVM writes the JSON String to the default output stream. For WebSpeed, write the JSON string to the WebSpeed-defined output stream (WEBSTREAM).

For more information about using ABL unnamed output streams, see the DEFINE STREAM statement reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces*. .

formatted

A CHARACTER expression that specifies the name of a stream to which the AVM writes the JSON string. If you specify the empty string (""), the AVM writes the JSON string to the default unnamed output stream. For WebSpeed, write the JSON string to the WebSpeed-defined output stream (WEBSTREAM).

encoding

An optional string indicating the Unicode Transformation Format of the JSON string to be written. Valid values for UTF are "UTF-8", "UTF-16", "UTF-16BE", "UTF-16LE", "UTF-32", "UTF-32BE",

and "UTF-32LE". If the value is the empty string (""), the Unknown value (?), or not given, the parameter defaults to "UTF-8".

WriteStream( ) method (JsonConstruct)

Serializes the JsonConstruct (JsonObject or JsonArray) and any JsonConstructs it references to a statically defined output stream.. This is a recursive process resulting in a JSON representation of the entire set of JsonObjects and JsonArrays. On successful execution the method returns TRUE.

Return type: LOGICAL

Access: PUBLIC ABSTRACT

Applies to: Progress.Json.ObjectModel.JsonConstruct class

Syntax

```
WriteStream(INPUT stream AS CHARACTER)

WriteStream(INPUT stream AS CHARACTER,
            INPUT formatted AS LOGICAL)

WriteStream(INPUT stream AS CHARACTER,
            INPUT formatted AS LOGICAL,
            INPUT encoding AS CHARACTER)
```

stream

A CHARACTER expression that specifies the name of a stream to which the AVM writes the JSON string. If you specify the empty string (""), the AVM writes the JSON string to the default unnamed output stream. For WebSpeed, write the JSON string to the WebSpeed-defined output stream (WEBSTREAM).

For more information about using ABL unnamed output streams, see the DEFINE STREAM statement reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces*.

formatted

An LOGICAL expression where TRUE directs the AVM to format the JSON string in a hierarchical manner using extra white space, carriage returns, and line feeds. The default value is FALSE.

If you specify the Unknown value (?), the method uses the default value of FALSE.

encoding

A CHARACTER expression indicating the Unicode Transformation Format of the JSON string to be written. Valid values for UTF are "UTF-8", "UTF-16", "UTF-16BE", "UTF-16LE", "UTF-32", "UTF-32BE", and "UTF-32LE". If the value is the empty string (""), the Unknown value (?), or not given, the parameter defaults to "UTF-8". If the value is any other value, an error is raised.

WriteStream( ) method (JsonObject)

Serializes the JsonObect and any JsonConstructs it references to a statically defined output stream.. This is a recursive process resulting in a JSON representation of the entire set of JsonObjects and JsonArrays. On successful execution the method returns TRUE.

Return type: LOGICAL

Access: PUBLIC

Applies to: Progress.Json.ObjectModel.JsonObject class

Syntax

```
WriteStream(INPUT stream AS CHARACTER)

WriteStream(INPUT stream AS CHARACTER,
            INPUT formatted AS LOGICAL)

WriteStream(INPUT stream AS CHARACTER,
            INPUT formatted AS LOGICAL,
            INPUT encoding AS CHARACTER)
```

stream

A CHARACTER expression that specifies the name of a stream to which the AVM writes the JSON string. If you specify the empty string (""), the AVM writes the JSON string to the default unnamed output stream. For WebSpeed, write the JSON string to the WebSpeed-defined output stream (WEBSTREAM).

For more information about using ABL unnamed output streams, see the DEFINE STREAM statement reference entry and the topics on alternate I/O sources in *OpenEdge Programming Interfaces*.

formatted

An LOGICAL expression where TRUE directs the AVM to format the JSON string in a hierarchical manner using extra white space, carriage returns, and line feeds. The default value is FALSE.

If you specify the Unknown value (?), the method uses the default value of FALSE.

encoding

A CHARACTER expression indicating the Unicode Transformation Format of the JSON string to be written. Valid values for UTF are "UTF-8", "UTF-16", "UTF-16BE", "UTF-16LE", "UTF-32", "UTF-32BE", and "UTF-32LE". If the value is the empty string (""), the Unknown value (?), or not given, the parameter defaults to "UTF-8". If the value is any other value, an error is raised.

WriteUnsignedLong( ) method

Write the unsigned 4-byte value of *i* to the current MemoryOutputStream object.

Note: This method is equivalent to the PUT-UNSIGNED-LONG statement, which applies to MEMPTR variables but not to MemoryOutputStream objects.

Return type: VOID

Access: PUBLIC

Applies to: Progress.IO.MemoryOutputStream class

Syntax

```
WriteUnsignedLong(INPUT i AS INT64)
```

WriteUnsignedShort( ) method

Write the unsigned 2-byte value of *i* to the current MemoryOutputStream object.

Note: This method is equivalent to the PUT-UNSIGNED-SHORT statement, which applies to MEMPTR variables but not to MemoryOutputStream objects.

Return type: VOID

Access: PUBLIC

Applies to: Progress.IO.MemoryOutputStream class

Syntax

```
WriteUnsignedShort(INPUT i AS INT64)
```

Xor( ) method
(Windows only; GUI for .NET only)

Performs a bitwise exclusive OR operation on the underlying values of the specified enumeration types and returns a new instance. You typically use this method to toggle a flag.

Return type: System.Enum class (from the .NET Framework)

Access: PUBLIC STATIC

Applies to: Progress.Util.EnumHelper class

Syntax

```
EnumHelper:Xor ( INPUT enum1 AS CLASS System.Enum,
                 INPUT enum2 AS CLASS System.Enum )
```

enum1

Static reference to an enumeration value, or a reference to an enumeration type instance.

enum2

Static reference to an enumeration value, or a reference to an enumeration type instance.

Specify a static reference to an enumeration value using the following syntax:

```
enum-type:enum-member
```

Where *enum-type* is the type name of the enumeration and *enum-member* is a member name of the enumeration. For example:

```
System.Windows.Forms.AnchorStyles:Bottom
```

The following example uses this method to toggle a flag:

```
USING System.Windows.Forms.* FROM ASSEMBLY.
USING Progress.Util.* FROM ASSEMBLY.

DEFINE VARIABLE rMyStyle AS System.Enum.

/* Adds Bottom and Right to the list */
rMyStyle = EnumHelper:Or( AnchorStyles:Bottom, AnchorStyles:Right ).
DISPLAY STRING( rMyStyle ) FORMAT "x(20)" SKIP.
/* Displays "Bottom, Right" */

/* Toggles a value */
rMyStyle = EnumHelper:Xor( rMyStyle, AnchorStyles:Right ).
DISPLAY STRING( rMyStyle ) FORMAT "x(20)" SKIP.
/* Displays "Bottom" */

rMyStyle = EnumHelper:Xor( rMyStyle, AnchorStyles:Right ).
DISPLAY STRING( rMyStyle ) FORMAT "x(20)" SKIP.
/* Displays "Bottom, Right" */
```

If you want to use the returned System.Enum instance as a different enumeration type, you must cast the instance to that specific type using the CAST function.

For information on the System.Enum class and .NET enumeration types, refer to the .NET Framework class library.

See also

CAST function, Or( ) method

10

Class Events Reference

Events are application notifications that occur as the result of run-time conditions. The ABL Virtual Machine (AVM) notifies your application of an event by executing ABL code that you define in a trigger or event handler, depending on the kind of event. ABL supports a number of built-in events that automatically result from various ABL, user, or other actions that affect an application.

You can also define your own events to notify your application of any useful condition. When you detect the occurrence of a given condition, you can notify your application of the event through an object associated with the event, which in turn executes a routine (event handler) that you have defined to respond to the event.

There are a number of factors that determine how ABL notifies your application of an event, the most important being the kind of object associated with the event. In ABL, events can be associated with the following types of objects:

- Handle-based objects
- Class-based objects

This section describes events that are associated with class-based objects (*class events*), including both ABL and .NET objects, and how your application can respond to them. The section also contains reference entries for built-in methods (*event methods*) that ABL provides to manage class events and reference entries for class events that ABL supports on built-in classes defined by OpenEdge. For more information on events associated with handle-based objects, see the Handle-based Object Events Reference.

The following descriptions refer to both compile-time and run-time behavior, which include features that the language supports and verifies at compile time and actions that result from using these features at run time. This section refers to ABL or the ABL compiler when describing compile-time features and refers to the AVM when describing run-time actions.

Class events

In ABL, class events are always defined as part of a class-based object definition, which can include one of the following object types:

- An ABL class or interface type
- A .NET class or interface type

ABL class events and .NET class events (also called *.NET events*) are very similar in concept, and OpenEdge allows you to manage both of them using the same ABL mechanism. Each class or interface can define any number of class events. Thus, class events are members of the class or interface for which they are defined, like methods, properties, and data members. Like any class member, a class event can be defined as either an instance or a static event, and an instance event can also be defined as abstract. A class event also has an access mode (private, package-private, protected, package-protected, or public), which determines the application scope that can respond to the event.

A class event has a signature, similar to a method, that can define parameters for passing data in response to event notification. An ABL application receives notice of a class event when the object that defines the event *publishes* it. In an ABL application, you can subscribe one or more ABL class methods or internal procedures as *event handlers*, which execute when a given event is published. These event handlers must all be defined with a signature that is compatible with the event that they are subscribed to handle.

Class events provide similar features for class-based objects that named events provide for procedure-based objects, but using different mechanisms. You publish a named event using the PUBLISH statement and manage named event handler subscriptions using the SUBSCRIBE statement and UNSUBSCRIBE statement. However, you publish a class event using the Publish( ) event method and manage class event handler subscriptions using the Subscribe( ) event method and Unsubscribe( ) event method. Among the differences between the two types of events, a named event is defined when it is published and without a strongly-typed signature, while a class event must be defined before it is published and with a strongly-typed signature.

Define class events

In ABL, you can define class events in a user-defined class or interface definition using the DEFINE EVENT statement. This statement defines all of the event options described in this section, including the signature for its event handlers.

In .NET, Common Language Specification (CLS)-compliant languages, such as C# or Visual Basic, have their own syntax to define events in a .NET class or interface. Similar to any other .NET class member, when you look up information about a .NET event in the .NET class library documentation, it typically describes the event member using syntax from one or more of these .NET languages. For example, C# provides the `event` keyword to declare an event in a class definition. In this way, the language-specific syntax for .NET events defines event options that are similar to the ABL DEFINE EVENT statement.

For ABL class events, you define the strongly-typed handler signature in the DEFINE EVENT statement. For .NET events, the handler signature is specified using a .NET class type known as a *delegate* that is referenced in the event definition. Thus, using the same delegate type, .NET objects can conveniently define multiple events whose handlers use the same signature. For an ABL class event, you can define either an ABL method signature or you can reference a .NET delegate type to define a .NET signature for the event. Referencing a .NET delegate also allows you to implement a .NET interface event or an inherited .NET abstract event in an ABL class.

You can define any valid ABL method signature for an ABL class event. However, the mechanics of event handler execution might limit your choices in practice. Also, ABL requires any .NET delegate that you use to conform to .NET conventions for event handlers.

Define event handlers

To use class events in your application, you must create one or more event handlers. A class event handler can be any class method or internal procedure whose signature is compatible with the signature defined for a given ABL or .NET class event. However, verification of the event handler signature occurs at a different point for method event handlers (at handler subscription compile time) than for internal procedure event handlers (at event publish run time). Note that all class event signatures include a VOID return type; therefore, any event handler method must also have a VOID return type.

For ABL class events, including ABL class events that implement inherited .NET abstract events, if you raise ERROR or throw an error object, the error is raised on the statement that executes the Publish( ) event method on the event as if you had called the event handler directly. For information on errors raised in handlers for .NET events, see Define handlers for .NET events.

Define multiple handlers for an event

If you need to subscribe more than one event handler for an ABL class event, when you publish the event, any OUTPUT or INPUT-OUTPUT parameter you define returns the value set by the final event handler to execute. In addition, any INPUT-OUTPUT parameter you define passes the value set by one event handler as input to the next event handler to execute. A similar effect occurs if you pass a handle or object reference as an INPUT parameter and your event handlers set the value of a public data member on the handle or class-based object. The returned value of the input object data element is the value set by the final handler to execute for the event. In addition, the order of execution for multiple handlers subscribed to a single event is not guaranteed, making it difficult or impossible to know exactly how parameter values returned from the `Publish( )` method have been set. So, if you use more than one handler for a given event, you need to use caution in defining and using the parameters for that event.

Also, if you raise ERROR or throw an error object from any one of the handlers for the ABL class event, any event handler that has not already run when the error is raised does not run. Therefore, if you want all handlers to run for an event, regardless of error conditions, you must resolve all conditions within each handler and allow the handler to return successfully.

Define handlers for .NET events

If you use a .NET delegate to define an ABL class event, or if you are defining an ABL method or internal procedure as a handler for a .NET event, you must define your event handler signature to match the delegate specified for the event. (In .NET, you implement event handlers by defining derivations of the specified delegate type.) For both ABL class events and .NET events, ABL only supports delegates whose signatures conform to the Microsoft .NET convention for event handler signatures. This signature convention provides for a VOID return type and two INPUT parameters, where the first parameter is a reference to a .NET `System.Object` that published the event, and the second parameter is a reference to a .NET `System.EventArgs` (or a derived class) that provides event arguments as public members of the class.

Syntax

The signature for any ABL event handler defined for a .NET delegate must conform to this general syntax:

```
VOID EventHandlerName( INPUT sender AS CLASS System.Object,
                       INPUT args AS CLASS EventArgsClass )
```

EventHandlerName

The name of your class-based method or internal procedure.

sender

Object reference to the .NET class instance that published the event.

args

Object reference to an event arguments class that contains public properties that provide arguments for the event. Note that for a .NET event, *args* references an object that is created by .NET and added to the ABL session object chain when the event handler executes. Like any other locally scoped object reference, if you do not save the value before the event handler terminates, ABL automatically garbage collects this object at some point after the event handler returns.

EventArgsClass

The class type name of the event arguments class. This is always `System.EventArgs` or one of its derived classes. Thus, `System.EventArgs` is the base class for all event arguments classes that an event handler can receive through this INPUT parameter.

To identify the exact handler signature for a given .NET delegate, you must look up the delegate in the appropriate class library documentation or use the Class Browser of Progress Developer Studio for OpenEdge to inspect the class. For .NET events supported on the built-in .NET classes provided by OpenEdge and described in the Class, Interface, and Enumeration Reference, each event reference entry described further on in this section indicates the delegate type associated with the event along with its matching event handler signature. Otherwise, to locate the delegate in .NET class library documentation, find the event you want to handle in the documentation for the class that publishes the event. The event definition includes a reference to its delegate type. The documentation for the specified delegate shows the .NET signature you must use for your event handler.

If you do not trap and handle an error raised within a handler for a .NET event, whether it is raised as the ERROR condition or thrown as an error object, the AVM does not throw an `Exception` back to .NET, but displays an error message on the default output device and continues processing as if no error had occurred. So, unlike handlers for ABL class events, if you subscribe multiple handlers for a .NET event, all the handlers execute regardless if one or more of them raises an error.

Use class events

For any ABL class event, used by itself or to implement an interface or inherited abstract event, you must follow these general steps:

1. Define the event using the 'DEFINE EVENT statement' in *Develop Object-oriented ABL Applications*.
2. Subscribe one or more handlers to the event using the 'Subscribe( ) event method' in *ABL Reference*. You can subscribe handlers to an event anywhere the event is accessible in an application.
3. Publish the event for a given condition using the 'Publish( ) event method' in *ABL Reference*, which runs all subscribed event handlers for the event. You can publish the event only from within the class definition that implements a non-abstract definition for the event.
4. Process any values returned from the Publish( ) method parameters.

At any point after subscribing event handlers, you can optionally unsubscribe any event handlers you no longer need using the 'Unsubscribe( ) event method' in *ABL Reference*. Thus, like named events for procedures, class events allow you to dynamically define and undefine alternate routines that execute in response to the event.

Note that similar to ABL class events, only the class that implements a non-abstract definition for a .NET event can publish it. This includes any ABL class that derives from and implements a .NET abstract class defining abstract events or that implements a .NET interface defining event prototypes.

However by convention, a .NET class that defines an event also typically defines a corresponding protected `On`*`EventName`*`( )` method that allows a derived class to publish the event, where *`EventName`* specifies the name of the .NET event and the signature defines a `System.EventArgs` (or derivation) as input. If the `On`*`EventName`*`( )` method is overrideable (`virtual` in C#), you can also override this method in an ABL derived class. If you do override this method, you must also invoke the super class implementation of `On`*`EventName`*`( )` from within the overriding method (using the 'SUPER system reference' in *Develop Object-oriented ABL Applications*) in order to ensure that all subscribed .NET delegates also respond to the event.

For more information on overriding .NET methods, see the 'METHOD statement' in *ABL Reference*.

To allow .NET events from .NET form and control objects to be published as part of the OpenEdge GUI for .NET, you must block to display the .NET forms and allow your subscribed event handlers to execute in response to any published events. To enable this .NET event handling, ABL provides WAIT-FOR statement syntax that blocks for both GUI for .NET events and ABL handle-based object events.

For more information, see the 'WAIT-FOR statement (.NET and ABL)' in *ABL Reference*.

Otherwise, you can work with .NET events in exactly the same way as ABL class events.

Events and event methods reference

The following reference entries document both the events of built-in ABL classes and the built-in event methods for working with all class events. The ABL built-in classes also include a set of .NET objects defined by OpenEdge. For more information on all ABL built-in classes, see the Class, Interface, and Enumeration Reference. For information on class events defined for all other .NET objects, including the OpenEdge Ultra Controls for .NET, see the Microsoft, Infragistics, or other third-party documentation provided for a given .NET object.

The reference entries for events that follow appear in alphabetical order by event name. Each reference entry includes the syntax required to define the signature for the event handler as a class-based method, with *`EventHandlerName`* used to indicate a name that you specify for the event handler. If you define the event handler as an internal procedure, define the same number of parameters in order by parameter mode and data type. If the event is a .NET event, the entry description also lists the name of the .NET delegate type defined for the event.

AfterFetchNextBatch event
(Windows only; GUI for .NET only)

This .NET event fires after the `BindingSource` object fetches the next batch of records when either the `Batching` property is set to true or the `MaxDataGuess` property is non-zero.

Access: PUBLIC

Applies to: Progress.Data.BindingSource class

Syntax

The delegate for this event defines the following event handler signature:

```
VOID EventHandlerName
  (
     INPUT sender AS CLASS System.Object,
     INPUT args AS CLASS System.EventArgs
  ).
```

EventHandlerName

The name of the event handler.

sender

Object reference to the `BindingSource` object instance that published the event.

args

Object reference to a `System.EventArgs` object instance containing arguments for the event.

When the event fires, there may be action that needs to be taken, such as calling control-specific methods or properties, to govern the control's behavior during this record fetching operation.

See also

Batching property, MaxDataGuess property

BeforeFetchNextBatch event
(Windows only; GUI for .NET only)

This .NET event fires before the `BindingSource` object fetches the next batch of records when either the `Batching` property is set to true or the `MaxDataGuess` property is non-zero.

Access: PUBLIC

Applies to: Progress.Data.BindingSource class

Syntax

The delegate for this event defines the following event handler signature:

```
VOID EventHandlerName
  (
     INPUT sender AS CLASS System.Object,
     INPUT args AS CLASS System.EventArgs
  ).
```

EventHandlerName

The name of the event handler.

sender

Object reference to the `BindingSource` object instance that published the event.

args

Object reference to a `System.EventArgs` object instance containing arguments for the event.

When the event fires, there may be action that needs to be taken, such as calling control-specific methods or properties, to govern the control's behavior during this record fetching operation.

See also

Batching property, MaxDataGuess property

CancelCreateRow event
(Windows only; GUI for .NET only)

The .NET event published when some user action cancels a create row operation in the bound .NET control (for example, pressing **ESCAPE** in a new empty row).

Access: PUBLIC

Applies to: Progress.Data.BindingSource class

Syntax

The delegate for this event defines the following event handler signature:

```
VOID EventHandlerName
  (
     INPUT sender AS CLASS System.Object,
     INPUT args AS CLASS Progress.Data.CancelCreateRowEventArgs
  ).
```

EventHandlerName

The name of the event handler.

sender

Object reference to the BindingSource object instance that published the event.

args

Object reference to a `CancelCreateRowEventArgs` object instance containing arguments for the event.

When this event occurs, delete the previously created row and the corresponding record in the result set. Otherwise, the data in the control will not be synchronized with its data source.

See also

CreateRow event, Progress.Data.CancelCreateRowEventArgs class

---

CreateRow event
(Windows only; GUI for .NET only)

The .NET event published when some user action initiates a create row operation in the bound .NET control (for example, clicking in a new empty row at the bottom of a control).

Access: PUBLIC

Applies to: Progress.Data.BindingSource class

Syntax

The delegate for this event defines the following event handler signature:

```
VOID EventHandlerName
  (
     INPUT sender AS CLASS System.Object,
     INPUT args AS CLASS Progress.Data.CreateRowEventArgs
  ).
```

EventHandlerName

The name of the event handler.

sender

Object reference to the `BindingSource` object instance that published the event.

args

Object reference to a `CreateRowEventArgs` object instance containing arguments for the event.

When this event occurs, create a new record in the ABL data source object and the current result set. You can also set any initial values you want presented to the user.

If the record was successfully created, set the `Created` property for the `CreateRowEventArgs` object to TRUE (the default value). If the record was not successfully created, set the `Created` property to FALSE.

Caution: After creating a row, do not reopen the query or invoke the `Refresh( )` method as this might produce unexpected results.

See also

CancelCreateRow event, Created property, Progress.Data.CreateRowEventArgs class

OffEnd event (.NET)
(Windows only; GUI for .NET only)

The .NET event published when record batching is enabled (that is, the `BindingSource` object's `Batching` property is TRUE) and some user action in the bound .NET control reaches the last row of the current result set. Use this event to retrieve the next batch of records.

You can use this event when the BindingSource object is bound to a query. If bound to a ProDataSet object, this event applies only to the top-level table displayed in the .NET control.

Access: PUBLIC

Applies to: Progress.Data.BindingSource class

Syntax

The delegate for this event defines the following event handler signature:

```
VOID EventHandlerName
  (
     INPUT sender AS CLASS System.Object,
     INPUT args AS CLASS Progress.Data.OffEndEventArgs
  ).
```

EventHandlerName

The name of the event handler.

sender

Object reference to the `BindingSource` object instance that published the event.

args

Object reference to an `OffEndEventArgs` object instance containing arguments for the event.

Each time you retrieve a batch of records, you must:

- Add the records to the current result set.
- Set the `RowsAdded` property for the `OffEndEventArgs` object to the number of records added to the result set (that is, the number of records retrieved in the most recent batch).

Once you have retrieved all of the records, set the `Batching` property to FALSE.

See also

Batching property, Progress.Data.OffEndEventArgs class, RowsAdded property

---

PositionChanged event
(Windows only; GUI for .NET only)

The .NET event published when the value of the `Position` property (which specifies the position of the current row in the bound .NET control) changes, based either on some user action in the .NET control or a programmatic action. For example:

- The user clicks on a different row in a control.
- The user clicks on a **Next** button and the event handler for the button increments the `Position` property value.

Note: When you click on a child row, the `Position` property is set to the position of the corresponding top-level parent row.

Access: PUBLIC

Applies to: Progress.Data.BindingSource class

Syntax

The delegate for this event defines the following event handler signature:

```
VOID EventHandlerName
  (
     INPUT sender AS CLASS System.Object,
     INPUT args AS CLASS System.EventArgs
  ).
```

EventHandlerName

The name of the event handler.

sender

Object reference to the `BindingSource` object instance that published the event.

args

Object reference to a .NET `System.EventArgs` object instance containing arguments for the event. For information on the public class members inherited from the `System.EventArgs` class, refer to the .NET Framework class library.

Whenever the value of the `Position` property changes, the `BindingSource` object automatically synchronizes the buffer in the bound ABL data source object to correspond to the selected row in the bound .NET control.

Note: Generally, you should change the `Position` property, rather than navigating or repositioning the query associated with the bound ABL data source object. For more information about synchronizing data, see *Use GUI for .NET Objects in ABL Applications*.

See also

Position property (BindingSource)

Publish( ) event method

Publishes an ABL class event defined in the current class definition. Publishing an event causes any event handlers subscribed to the event to execute.

Return type: VOID

Access: PRIVATE

Applies to: ABL class events

Syntax

```
[ THIS-OBJECT : | class-type-name : ]
  event-name : Publish ( [parameter[ , parameter]...] ) [ NO-ERROR ]
```

[THIS-OBJECT : | *class-type-name* :]

If *event-name* is a reserved keyword, you must prefix an instance *event-name* with THIS-OBJECT or a static *event-name* with the class type name of the current class definition. Otherwise use of these qualifiers is optional.

event-name

The name of a non-abstract event defined as part of the current class definition. At compile time, ABL verifies that *event-name* specifies an event whose DEFINE EVENT statement appears in the current class definition. This can also be the name of an inherited .NET abstract event that is implemented in the current ABL class.

([*parameter* [, *parameter*] ...])

Specifies zero or more parameters as defined for the event signature that you must pass to the Publish( ) method. The parameters you pass must match the parameters defined in the corresponding DEFINE EVENT statement with respect to number, data type, and mode, exactly as if you were calling a method defined with these parameters. Otherwise, ABL raises a compile-time error. Note that even if the method compiles, the AVM can raise a run-time error if an argument passed to a parameter has an ambiguous type during compilation that turns out to be incompatible at run time.

The Publish( ) method passes the same parameters to each event handler subscribed to the event. Note that any parameter results represent values returned from the last event handler to execute. However, the order of execution for event handlers is not guaranteed. Therefore, if you subscribe to multiple event handlers, you cannot be certain what event handler has returned the parameter values from the Publish( ) method.

Caution: Any values passed as INPUT-OUTPUT to a given event handler become input to the next event handler to execute for a given event. Because the order of handler execution for multiple event handlers is not guaranteed, you cannot be certain of the input values passed as INPUT-OUTPUT to any given event handler.

For more information on the syntax and requirements for passing each *parameter*, and on the behavior of parameters passed to the Publish( ) method, see the Parameter passing syntax reference entry.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

For an ABL class event, if an event handler invoked by this method raises an error, the error is raised on this statement, and any event handlers that have not yet executed for the event do not execute. In this case, the error behavior and messages reflect the event handler that generated the error as if you called the handler directly. Note also that any STOP or QUIT condition is handled as if you called the handler directly.

Note also, if *event-name* is an ABL event that implements a .NET abstract event, and you invoke a .NET method within your handler that throws an exception back to .NET, .NET generates a System.ApplicationException that it throws back to the .NET method, which determines the result that you can manage in your handler.

You can only invoke the `Publish( )` method on an event from within a class definition that defines and implements the event, regardless of the event's access mode. (You cannot invoke `Publish( )` on an abstract event.) Thus, you can publish the event within any method, constructor, destructor, property accessor, or trigger that is defined within the class that also includes the DEFINE EVENT statement that implements the event.

After the `Publish( )` statement executes, the value of the RETURN-VALUE function reflects the last RETURN statement executed (if any) by all event handlers subscribed to *event-name*. However, because the order of handler execution is not guaranteed, if you subscribe more than one event handler that executes `RETURN`, you might not know which handler set the value for the `RETURN-VALUE` function.

See also

NO-ERROR option, PUBLISH statement, Subscribe( ) event method, Unsubscribe( ) event method

---

SortRequest event
(Windows only; GUI for .NET only)

The .NET event published when some user action initiates a sort operation in the bound .NET control (for example, clicking on a column header in a grid control).

You can use this event when the BindingSource object is bound to a query or a buffer. If bound to a ProDataSet object, this event applies only to the top-level table displayed in the .NET control.

Access: PUBLIC

Applies to: Progress.Data.BindingSource class

Syntax

The delegate for this event defines the following event handler signature:

```
VOID EventHandlerName
  (
     INPUT sender AS CLASS System.Object,
     INPUT args AS CLASS Progress.Data.SortRequestEventArgs
  ).
```

`EventHandlerName`

The name of the event handler.

`sender`

Object reference to the `BindingSource` object instance that published the event.

`args`

Object reference to a `SortRequestEventArgs` object instance containing arguments for the event.

When this event occurs, the event handler must reopen the query associated with the ABL data source object with a modified sort criteria based on the values specified in the `SortRequestEventArgs` object.

This event occurs only for .NET controls that rely on their data source for sorting.

See also

Progress.Data.SortRequestEventArgs class

Subscribe( ) event method

Subscribes an event handler method or procedure to an ABL or .NET class event.

Return type: VOID

Access: PUBLIC

Applies to: ABL or .NET class events

Syntax

```
[ publisher : ] event-name:Subscribe
  ( [ subscriber : ] handler-method ) [ NO-ERROR ]
```

```
[ publisher : ] event-name:Subscribe
  ( [ subscriber-handle , ] handler-procedure ) [ NO-ERROR ]
```

[*publisher* **:]**

The object reference to an ABL or .NET object or the type name of an ABL or .NET class that can publish the instance or static event, respectively. If you do not specify a *publisher*, it defaults to either THIS-OBJECT when *event-name* identifies an instance event defined in the current class instance, or to the class type name of the current class definition when *event-name* identifies a static event defined in the current class hierarchy.

event-name

The name of an ABL or .NET event that *publisher* can publish. At compile time, ABL verifies that *event-name* is an accessible member of the class referenced by *publisher*. The event can be an abstract event.

[*subscriber* **:]** *handler-method*

An ABL class-based method that you want to subscribe as a handler for *event-name*, where *handler-method* is the unquoted name of a pure ABL instance or static method. If specified, *subscriber* can be either the object reference to an ABL class instance where *handler-method* is defined, or the type name of an ABL class where the static *handler-method* is defined. If you do not specify a *subscriber*, it defaults to either THIS-OBJECT when *handler-method* identifies an instance method defined in the current class hierarchy, or to the class type name of the current class definition when *handler-method* identifies a static method defined in the current class hierarchy.

Note: The specified *handler-method* cannot be a .NET method.

Note that the access mode (PUBLIC, PACKAGE-PROTECTED, PROTECTED, PACKAGE-PRIVATE, or PRIVATE) of the *handler-method* definition does not matter to the *publisher*. For example, you can subscribe a PRIVATE method of the subscribing class instance as a handler to an event defined and published by yet another class instance.

The signature of *handler-method* does not have to be identical with the signature defined for *event-name*, but it must be run-time compatible. This includes an overloaded method as long as ABL can identify the signature for a unique overloading that is compatible with the event signature.

Thus, ABL raises a compile-time error if *handler-method*:

- Is not accessible to the context where this Subscribe( ) method executes
- Does not have a run-time compatible signature
- Does not have a unique overloading that is compatible with the event signature

[*subscriber-handle* **,** **]** *handler-procedure*

A procedure that you want to subscribe as a handler for *event-name*, where *handler-procedure* is a character expression that evaluates to the name of an internal procedure. If specified, *subscriber-handle* is a handle to a persistent procedure, or any other external procedure on the call stack, where the internal procedure is defined. Otherwise, *subscriber-handle* defaults to THIS-PROCEDURE.

At compile-time, ABL checks this Subscribe( ) method call only to verify that any specified *subscriber-handle* is a HANDLE data type and that *handler-procedure* is specified by a character expression. Otherwise, the AVM raises a run-time error if the procedure context specified by *subscriber-handle* is not accessible or it does not define *handler-procedure* as an internal procedure.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

See also

NO-ERROR option, Publish( ) event method, SUBSCRIBE statement, Unsubscribe( ) event method

Unsubscribe( ) event method

Removes a method or procedure as a handler for an ABL or .NET class event.

Return type: VOID

Access: PUBLIC

Applies to: ABL or .NET class events

Syntax

```
[ publisher : ] event-name:Unsubscribe
  ( [ subscriber : ] handler-method ) [ NO-ERROR ]
```

```
[ publisher : ] event-name:Unsubscribe
  ( [ subscriber-handle , ] handler-procedure ) [ NO-ERROR ]
```

[*publisher* :]

The object reference to an ABL or .NET object or the type name of an ABL or .NET class that can publish the instance or static event, respectively. If you do not specify a *publisher*, it defaults to either THIS-OBJECT when *event-name* identifies an instance event defined in the current class instance, or to the class type name of the current class definition when *event-name* identifies a static event defined in the current class hierarchy.

event-name

The name of an ABL or .NET event that *publisher* can publish. At compile time, ABL verifies that *event-name* is an accessible member of the class referenced by *publisher*. The event can be an abstract event.

[*subscriber* :] *handler-method*

An ABL class-based method that you want to remove as a handler for *event-name*, where *handler-method* is the unquoted name of the instance or static method. If specified, *subscriber* can be either the object reference to an ABL class instance where *handler-method* is defined, or the type name of an ABL class where the static *handler-method* is defined. If you do not specify a *subscriber*, it defaults to THIS-OBJECT when *handler-method* identifies an instance method defined in the current class hierarchy, or to the class type name of the current class definition when *handler-method* identifies a static method defined in the current class hierarchy.

Note that the access mode (PUBLIC, PACKAGE-PROTECTED, PROTECTED, PACKAGE-PRIVATE, or PRIVATE) of the *handler-method* definition does not matter to the *publisher*. For example, you can remove a PRIVATE method of the unsubscribing class instance as a handler for an event defined and published by yet another class instance.

The signature of *handler-method* must be run-time compatible with the signature defined for *event-name*. This includes an overloaded method as long as ABL can identify the signature for a unique overloading that is compatible with the event signature.

Thus, ABL raises a compile-time error if *handler-method*:

- Is not accessible to the context where this `Unsubscribe( )` method executes
- Does not have a run-time compatible signature
- Does not have a unique overloading that is compatible with the event signature

[*subscriber-handle* ,] *handler-procedure*

A procedure that you want to remove as a handler for *event-name*, where *handler-procedure* is a character expression that evaluates to the name of an internal procedure. If specified, *subscriber-handle* is a handle to a persistent procedure, or other external procedure on the call stack, where the internal procedure is defined. Otherwise, *subscriber-handle* defaults to THIS-PROCEDURE.

At compile-time, ABL checks this Unsubscribe( ) method call only to verify that any specified *subscriber-handle* is a HANDLE data type and that *handler-procedure* is specified by a character expression. Otherwise, ABL raises a run-time error if the procedure context specified by *subscriber-handle* is not accessible or it does not define *handler-procedure* as an internal procedure.

NO-ERROR

The NO-ERROR option is used to prevent the statement from raising ERROR and displaying error messages.

The Unsubscribe( ) method cancels an event subscription by removing the specified method or internal procedure from the list of handlers for the specified *event-name*. If you do not specify a *publisher*, you can only invoke Unsubscribe( ) in an ABL class definition and on an ABL or .NET event that the ABL class defines or inherits. When you cancel an event handler subscription, ABL does not raise either a run-time error if the specified event handler is not currently on the subscription list for the specified event.

See also

NO-ERROR option, Publish( ) event method, Subscribe( ) event method, UNSUBSCRIBE statement

11

Keyword Index

The following table lists all keywords and built-in object names in ABL. The columns are as follows:

- **Keyword** — Specifies the full keyword or built-in object name.
- **Rsrv** — Indicates whether a keyword is reserved.
- **Minimum abbreviation** — Specifies the shortest abbreviation ABL recognizes for the keyword or name. If no abbreviation is specified, the keyword cannot be abbreviated.

Note: As a best practice avoid the use of abbreviated keywords. You can use the `COMPILE OPTIONS require-full-keywords` option to enforce this.

Keyword	Rsrv	Minimum abbreviation
+	✓	–
-	✓	–
.	✓	–
&ELSE	✓	–
&ELSEIF	✓	–
&ENDIF	✓	–

Keyword	Rsrv	Minimum abbreviation
&GLOBAL-DEFINE	✓	&GLOB
&IF	✓	–
&MESSAGE	✓	–
&SCOPED-DEFINE	✓	&SCOP
&THEN	✓	–
&UNDEFINE	✓	&UNDEF
&WEBSTREAM	✓	–
*	✓	–
/	✓	–
:	✓	–
?	✓	–
@	✓	–
[	✓	–
]	✓	–
^	✓	–
{&BATCH-MODE}	–	{&BATCH}
{&FILE-NAME}	–	–
{&LINE-NUMBER}	–	{&LINE-NUMBE}
{&OPSYS}	–	–
{&PROCESS-ARCHITECTURE}	–	–
{&SEQUENCE}	–	–
{&WINDOW-SYSTEM}	–	{&WINDOW-SYS}
'	✓	–
<	✓	–

Keyword	Rsrv	Minimum abbreviation
<=	–	–
<>	–	–
=	✓	–
>	✓	–
>=	–	–
ABSOLUTE	–	ABS
ABSTRACT	–	–
ACCELERATOR	–	–
ACCUM	✓	–
ACCUMULATE	✓	ACCUM
ACTIVE-FORM	✓	–
ACTIVE-WINDOW	✓	–
ADD	✓	–
ADD-BUFFER	–	–
ADD-CALC-COLUMN	–	–
ADD-COLUMNS-FROM	–	–
ADD-EVENTS-PROCEDURE	–	–
ADD-FIELDS-FROM	–	–
ADD-FIRST	–	–
ADD-INDEX-FIELD	–	–
ADD-LAST	–	–
ADD-LIKE-COLUMN	–	–
ADD-LIKE-FIELD	–	–
ADD-LIKE-INDEX	–	–
ADD-NEW-FIELD	–	–

Keyword	Rsrv	Minimum abbreviation
ADD-NEW-INDEX	–	–
ADD-SCHEMA-LOCATION	–	–
ADD-SUPER-PROCEDURE	–	–
ADM-DATA	–	–
ADVISE	–	–
AGGREGATE	–	–
ALERT-BOX	–	–
ALIAS	✓	–
ALL	✓	–
ALLOW-COLUMN-SEARCHING	–	–
ALLOW-PREV-DESERIALIZATION	–	–
ALLOW-REPLICATION	–	–
ALTER	✓	–
ALWAYS-ON-TOP	–	–
AMBIGUOUS	✓	AMBIG
ANALYZE	✓	ANALYZ
AND	✓	–
ANSI-ONLY	–	–
ANY	✓	–
ANYWHERE	–	–
APPEND	–	–
APPL-ALERT-BOXES	–	APPL-ALERT
APPL-CONTEXT-ID	–	–
APPLICATION	–	–
APPLY	✓	–

Keyword	Rsrv	Minimum abbreviation
APPSERVER-INFO	–	–
APPSERVER-PASSWORD	–	–
APPSERVER-USERID	–	–
ARRAY-MESSAGE	–	–
AS	✓	–
ASC	–	–
ASCENDING	✓	ASC
ASK-OVERWRITE	–	–
ASSEMBLY	–	–
ASSIGN	✓	–
ASYNCHRONOUS	✓	–
ASYNC-REQUEST-COUNT	–	–
ASYNC-REQUEST-HANDLE	–	–
AT	✓	–
ATTACHED-PAIRLIST	–	–
ATTR-SPACE	✓	ATTR
AUDIT-CONTROL	✓	–
AUDIT-ENABLED	–	–
AUDIT-EVENT-CONTEXT	–	–
AUDIT-POLICY	✓	–
AUTHENTICATION-FAILED	–	–
AUTHORIZATION	✓	–
AUTO-COMPLETION	–	AUTO-COMP
AUTO-ENDKEY	–	–
AUTO-END-KEY	–	–

Keyword	Rsrv	Minimum abbreviation
AUTO-GO	–	–
AUTO-INDENT	–	AUTO-IND
AUTOMATIC	–	–
AUTO-RESIZE	–	–
AUTO-RETURN	✓	AUTO-RET
AUTO-SYNCHRONIZE	–	–
AUTO-ZAP	–	AUTO-Z
AVAILABLE	✓	AVAIL
AVAILABLE-FORMATS	–	–
AVERAGE	–	AVE
AVG	–	–
BACKGROUND	✓	BACK
BACKWARDS	–	BACKWARD
BASE64-DECODE	–	–
BASE64-ENCODE	–	–
BASE-ADE	–	–
BASE-KEY	–	–
BATCH-MODE	–	BATCH
BATCH-SIZE	–	–
BEFORE-HIDE	✓	BEFORE-H
BEGIN-EVENT-GROUP	–	–
BEGINS	✓	–
BELL	✓	–
BETWEEN	✓	–
BGCOLOR	–	BGC

Keyword	Rsrv	Minimum abbreviation
BIG-ENDIAN	✓	–
BINARY	–	–
BIND	–	–
BIND-WHERE	–	–
BLANK	✓	–
BLOCK-ITERATION-DISPLAY	–	–
BLOCK-LEVEL	–	BLOCK-LEV
BORDER-BOTTOM-CHARS	–	BORDER-B
BORDER-BOTTOM-PIXELS	–	BORDER-BOTTOM-P
BORDER-LEFT-CHARS	–	BORDER-L
BORDER-LEFT-PIXELS	–	BORDER-LEFT-P
BORDER-RIGHT-CHARS	–	BORDER-R
BORDER-RIGHT-PIXELS	–	BORDER-RIGHT-P
BORDER-TOP-CHARS	–	BORDER-T
BORDER-TOP-PIXELS	–	BORDER-TOP-P
BOX	–	–
BOX-SELECTABLE	–	BOX-SELECT
BREAK	✓	–
BROWSE	–	–
BUFFER	–	–
BUFFER-CHARS	–	–
BUFFER-COMPARE	–	–
BUFFER-COPY	–	–
BUFFER-CREATE	–	–
BUFFER-DELETE	–	–

Keyword	Rsrv	Minimum abbreviation
BUFFER-FIELD	–	–
BUFFER-HANDLE	–	–
BUFFER-LINES	–	–
BUFFER-NAME	–	–
BUFFER-PARTITION-ID	–	–
BUFFER-RELEASE	–	–
BUFFER-VALUE	–	–
BUTTON	–	–
BUTTONS	–	BUTTON
BY	✓	–
BY-POINTER	–	–
BY-VARIANT-POINTER	–	–
CACHE	–	–
CACHE-SIZE	–	–
CALL	✓	–
CALL-NAME	–	–
CALL-TYPE	–	–
CANCEL-BREAK	–	–
CANCEL-BUTTON	–	–
CAN-CREATE	–	–
CAN-DELETE	–	–
CAN-DO	✓	–
CAN-DO-DOMAIN-SUPPORT	–	–
CAN-FIND	✓	–
CAN-QUERY	–	–

Keyword	Rsrv	Minimum abbreviation
CAN-READ	–	–
CAN-SET	–	–
CAN-WRITE	–	–
CAPS	–	–
CAREFUL-PAINT	–	–
CASE	✓	–
CASE-SENSITIVE	✓	CASE-SEN
CAST	✓	–
CATCH	–	–
CDECL	–	–
CENTERED	✓	CENTER
CHAINED	–	–
CHARACTER	–	CHAR
CHARACTER_LENGTH	–	–
CHARSET	–	–
CHECK	✓	–
CHECKED	–	–
CHOOSE	–	–
CHR	✓	–
CLASS	–	–
CLASS-TYPE	–	–
CLEAR	✓	–
CLEAR-APPL-CONTEXT	–	–
CLEAR-LOG	–	–
CLEAR-SELECTION	–	CLEAR-SELECT

Keyword	Rsrv	Minimum abbreviation
CLEAR-SORT-ARROWS	–	CLEAR-SORT-ARROW
CLIENT-CONNECTION-ID	–	–
CLIENT-PRINCIPAL	–	–
CLIENT-TTY	–	–
CLIENT-TYPE	–	–
CLIENT-WORKSTATION	–	–
CLIPBOARD	✓	–
CLOSE	–	–
CLOSE-LOG	–	–
CODE	–	–
CODEBASE-LOCATOR	✓	–
CODEPAGE	–	–
CODEPAGE-CONVERT	–	–
COLLATE	–	–
COL-OF	–	–
COLON	✓	–
COLON-ALIGNED	–	COLON-ALIGN
COLOR	✓	–
COLOR-TABLE	–	–
COLUMN	–	COL
COLUMN-BGCOLOR	–	–
COLUMN-DCOLOR	–	–
COLUMN-FGCOLOR	–	–
COLUMN-FONT	–	–
COLUMN-LABEL	✓	COLUMN-LAB

Keyword	Rsrv	Minimum abbreviation
COLUMN-MOVABLE	–	–
COLUMN-OF	–	–
COLUMN-PFCOLOR	–	–
COLUMN-READ-ONLY	–	–
COLUMN-RESIZABLE	–	–
COLUMNS	✓	–
COLUMN-SCROLLING	–	–
COMBO-BOX	–	–
COM-HANDLE	–	–
COMMAND	–	–
COMPARE	–	–
COMPARES	–	–
COMPILE	–	–
COMPILER	✓	–
COMPLETE	–	–
COM-SELF	–	–
CONFIG-NAME	–	–
CONNECT	–	–
CONNECTED	✓	–
CONSTRUCTOR	–	–
CONTAINS	–	–
CONTENTS	–	–
CONTEXT	–	–
CONTEXT-HELP	–	–
CONTEXT-HELP-FILE	–	–

Keyword	Rsrv	Minimum abbreviation
CONTEXT-HELP-ID	–	–
CONTEXT-POPUP	–	–
CONTROL	✓	–
CONTROL-BOX	–	–
CONTROL-FRAME	–	–
CONVERT	–	–
CONVERT-3D-COLORS	–	–
CONVERT-TO-OFFSET	–	CONVERT-TO-OFFS
COPY-DATASET	–	–
COPY-LOB	✓	–
COPY-SAX-ATTRIBUTES	–	–
COPY-TEMP-TABLE	–	–
COUNT	–	–
COUNT-OF	✓	–
CPCASE	–	–
CPCOLL	–	–
CPINTERNAL	–	–
CPLOG	–	–
CPPRINT	–	–
CPRCODEIN	–	–
CPRCODEOUT	–	–
CPSTREAM	–	–
CPTERM	–	–
CRC-VALUE	–	–
CREATE	✓	–

Keyword	Rsrv	Minimum abbreviation
CREATE-LIKE	–	–
CREATE-LIKE-SEQUENTIAL	–	–
CREATE-NODE-NAMESPACE	–	–
CREATE-RESULT-LIST-ENTRY	–	–
CREATE-TEST-FILE	–	–
CURRENT	✓	–
CURRENT_DATE	✓	–
CURRENT_DATE	–	–
CURRENT-CHANGED	✓	–
CURRENT-COLUMN	–	–
CURRENT-ENVIRONMENT	–	CURRENT-ENV
CURRENT-ITERATION	–	–
CURRENT-LANGUAGE	✓	CURRENT-LANG
CURRENT-QUERY	–	–
CURRENT-REQUEST-INFO	–	–
CURRENT-RESPONSE-INFO	–	–
CURRENT-RESULT-ROW	–	–
CURRENT-ROW-MODIFIED	–	–
CURRENT-VALUE	–	–
CURRENT-WINDOW	✓	–
CURSOR	✓	CURS
CURSOR-CHAR	–	–
CURSOR-LINE	–	–
CURSOR-OFFSET	–	–
DATABASE	✓	–

Keyword	Rsrv	Minimum abbreviation
DATA-BIND	–	–
DATA-ENTRY-RETURN	–	DATA-ENTRY-RET
DATA-RELATION	✓	DATA-REL
DATASERVERS	✓	–
DATASET	✓	–
DATASET-HANDLE	✓	–
DATA-SOURCE	✓	–
DATA-SOURCE-COMPLETE-MAP	–	–
DATA-SOURCE-MODIFIED	–	–
DATA-SOURCE-ROWID	–	–
DATA-TYPE	–	DATA-T
DATE	–	–
DATE-FORMAT	–	DATE-F
DAY	–	–
DBCODEPAGE	✓	–
DBCOLLATION	✓	–
DBNAME	✓	–
DBPARAM	✓	–
DB-CONTEXT	–	–
DB-REFERENCES	–	–
DBRESTRICTIONS	✓	DBREST
DBTASKID	✓	–
DBTYPE	✓	–
DBVERSION	✓	DBVERS

Keyword	Rsrv	Minimum abbreviation
DCOLOR	–	–
DDE	✓	–
DDE-ERROR	–	–
DDE-ID	–	DDE-I
DDE-ITEM	–	–
DDE-NAME	–	–
DDE-TOPIC	–	–
DEBLANK	✓	–
DEBUG	–	DEBU
DEBUG-ALERT	–	–
DEBUGGER	✓	–
DEBUG-LIST	✓	–
DECIMAL	–	DEC
DECIMALS	✓	–
DECLARE	✓	–
DECLARE-NAMESPACE	–	–
DECRYPT	–	–
DEFAULT	✓	–
DEFAULT-BUFFER-HANDLE	–	–
DEFAULT-BUTTON	–	DEFAUT-B
DEFAULT-COMMIT	–	–
DEFAULT-EXTENSION	–	DEFAULT-EX
DEFAULT-NOXLATE	✓	DEFAULT-NOXL
DEFAULT-VALUE	–	–
DEFAULT-WINDOW	✓	–

Keyword	Rsrv	Minimum abbreviation
DEFINE	✓	DEF
DEFINED	–	–
DEFINE-USER-EVENT-MANAGER	–	–
DELEGATE	–	–
DELETE	✓	DEL
DELETE PROCEDURE	–	–
DELETE-CHAR	–	–
DELETE-CURRENT-ROW	–	–
DELETE-LINE	–	–
DELETE-RESULT-LIST-ENTRY	–	–
DELETE-SELECTED-ROW	–	–
DELETE-SELECTED-ROWS	–	–
DELIMITER	✓	–
DESC	–	–
DESCENDING	✓	DESC
DESELECT-FOCUSED-ROW	–	–
DESELECTION	–	–
DESELECT-ROWS	–	–
DESELECT-SELECTED-ROW	–	–
DESTRUCTOR	–	–
DIALOG-BOX	–	–
DICTIONARY	✓	DICT
DIR	–	–
DISABLE	✓	–
DISABLE-AUTO-ZAP	✓	–

Keyword	Rsrv	Minimum abbreviation
DISABLED	–	–
DISABLE-DUMP-TRIGGERS	–	–
DISABLE-LOAD-TRIGGERS	–	–
DISCONNECT	✓	DISCON
DISP	✓	–
DISPLAY	✓	DISP
DISPLAY-MESSAGE	–	–
DISPLAY-TYPE	–	DISPLAY-T
DISTINCT	✓	–
DO	✓	–
DOMAIN-DESCRIPTION	–	–
DOMAIN-NAME	–	–
DOMAIN-TYPE	–	–
DOS	✓	–
DOUBLE	–	–
DOWN	✓	–
DRAG-ENABLED	–	–
DROP	✓	–
DROP-DOWN	–	–
DROP-DOWN-LIST	–	–
DROP-FILE-NOTIFY	–	–
DROP-TARGET	–	–
ds-close-cursor	–	–
DSLOG-MANAGER	–	–
DUMP	–	–

Keyword	Rsrv	Minimum abbreviation
DYNAMIC	–	–
DYNAMIC-ENUM	✓	–
DYNAMIC-FUNCTION	✓	–
DYNAMIC-INVOKE	✓	–
EACH	✓	–
ECHO	–	–
EDGE-CHARS	–	EDGE
EDGE-PIXELS	–	EDGE-P
EDIT-CAN-PASTE	–	–
EDIT-CAN-UNDO	–	–
EDIT-CLEAR	–	–
EDIT-COPY	–	–
EDIT-CUT	–	–
EDITING	✓	–
EDITOR	–	–
EDIT-PASTE	–	–
EDIT-UNDO	–	–
ELSE	✓	–
EMPTY	–	–
EMPTY-TEMP-TABLE	–	–
ENABLE	✓	–
ENABLED-FIELDS	–	–
ENCODE	✓	–
ENCODE-DOMAIN-ACCESS-CODE	–	–
ENCRYPT	–	–

Keyword	Rsrv	Minimum abbreviation
ENCRYPT-AUDIT-MAC-KEY	–	–
ENCRYPTION-SALT	–	–
END	✓	–
END-DOCUMENT	–	–
END-ELEMENT	–	–
END-EVENT-GROUP	–	–
END-FILE-DROP	–	–
ENDKEY	–	–
END-KEY	–	–
END-MOVE	–	–
END-RESIZE	–	–
END-ROW-RESIZE	–	–
END-USER-PROMPT	–	–
ENTERED	–	–
ENTITY-EXPANSION-LIMIT	–	–
ENTRY	✓	–
ENUM	–	–
EQ	–	–
ERROR	–	–
ERROR-COLUMN	–	ERROR-COL
ERROR-OBJECT	–	–
ERROR-ROW	–	–
ERROR-STACK-TRACE	–	–
ERROR-STATUS	✓	ERROR-STAT
ESCAPE	✓	–

Keyword	Rsrv	Minimum abbreviation
ETIME	✓	–
EVENT	–	–
EVENT-GROUP-ID	–	–
EVENT-HANDLER	–	–
EVENT-HANDLER-CONTEXT	–	–
EVENT-PROCEDURE	✓	–
EVENT-PROCEDURE-CONTEXT	–	–
EVENTS	–	–
EVENT-TYPE	–	EVENT-T
EXCEPT	✓	–
EXCLUSIVE-ID	–	–
EXCLUSIVE-LOCK	✓	EXCLUSIVE
EXCLUSIVE-WEB-USER	–	–
EXECUTE	–	–
EXISTS	✓	–
EXIT-CODE	–	–
EXP	–	–
EXPAND	–	–
EXPANDABLE	–	–
EXPLICIT	–	–
EXPORT	✓	–
EXPORT-PRINCIPAL	–	–
EXTENDED	–	–
EXTENT	–	–
EXTERNAL	–	–

Keyword	Rsrv	Minimum abbreviation
FALSE	✓	–
FETCH	✓	–
FETCH-SELECTED-ROW	–	–
FGCOLOR	–	FGC
FIELD	✓	–
FIELDS	✓	FIELD
FILE	–	–
FILE-CREATE-DATE	–	–
FILE-CREATE-TIME	–	–
FILE-INFO	✓	–
FILE-MOD-DATE	–	–
FILE-MOD-TIME	–	–
FILENAME	–	–
FILE-NAME	–	–
FILE-OFFSET	–	FILE-OFF
FILE-SIZE	–	–
FILE-TYPE	–	–
FILL	✓	–
FILLED	–	–
FILL-IN	–	–
FILTERS	–	–
FINAL	–	–
FINALLY	–	–
FIND	✓	–
FIND-BY-ROWID	–	–

Keyword	Rsrv	Minimum abbreviation
FIND-CASE-SENSITIVE	✓	–
FIND-CURRENT	–	–
FINDER	–	–
FIND-FIRST	–	–
FIND-GLOBAL	✓	–
FIND-LAST	–	–
FIND-NEXT-OCCURRENCE	✓	–
FIND-PREV-OCCURRENCE	✓	–
FIND-SELECT	✓	–
FIND-UNIQUE	–	–
FIND-WRAP-AROUND	✓	–
FIRST	✓	–
FIRST-ASYNCH-REQUEST	–	–
FIRST-CHILD	–	–
FIRST-COLUMN	–	–
FIRST-FORM	–	–
FIRST-OBJECT	–	–
FIRST-OF	✓	–
FIRST-PROCEDURE	–	FIRST-PROC
FIRST-SERVER	–	–
FIRST-TAB-ITEM	–	FIRST-TAB-I
FIT-LAST-COLUMN	–	–
FIXED-ONLY	–	–
FLAT-BUTTON	–	–
FLOAT	–	–

Keyword	Rsrv	Minimum abbreviation
FOCUS	✓	–
FOCUSED-ROW	–	–
FOCUSED-ROW-SELECTED	–	–
FONT	✓	–
FONT-TABLE	–	–
FOR	✓	–
FORCE-FILE	–	–
FOREGROUND	–	FORE
FORM	✓	–
FORM INPUT	–	–
FORMAT	✓	FORM
FORMATTED	–	FORMATTE
FORM-LONG-INPUT	–	–
FORWARD	–	–
FORWARDS	–	FORWARD
FRAGMENT	–	FRAGMEN
FRAME	✓	FRAM
FRAME-COL	✓	–
FRAME-DB	✓	–
FRAME-DOWN	✓	–
FRAME-FIELD	✓	–
FRAME-FILE	✓	–
FRAME-INDEX	✓	FRAME-INDE
FRAME-LINE	✓	–

Keyword	Rsrv	Minimum abbreviation
FRAME-NAME	✓	–
FRAME-ROW	✓	–
FRAME-SPACING	–	FRAME-SPA
FRAME-VALUE	✓	FRAME-VAL
FRAME-X	–	–
FRAME-Y	–	–
FREQUENCY	–	–
FROM	✓	–
FROM-CHARS	✓	FROM-C
FROM-CURRENT	–	FROM-CUR
FROM-PIXELS	✓	FROM-P
FULL-HEIGHT-CHARS	–	FULL-HEIGHT
FULL-HEIGHT-PIXELS	–	FULL-HEIGHT-P
FULL-PATHNAME	–	FULL-PATHN
FULL-WIDTH-CHARS	–	FULL-WIDTH
FULL-WIDTH-PIXELS	–	FULL-WIDTH-P
FUNCTION	–	–
FUNCTION-CALL-TYPE	✓	–
GATEWAYS	✓	GATEWAY
GE	–	–
GENERATE-PBE-KEY	–	–
GENERATE-PBE-SALT	–	–
GENERATE-RANDOM-KEY	–	–
GENERATE-UUID	–	–
GET	–	–

Keyword	Rsrv	Minimum abbreviation
GET-ATTR-CALL-TYPE	✓	–
GET-ATTRIBUTE-NODE	–	–
GET-BINARY-DATA	–	–
GET-BLUE-VALUE	–	GET-BLUE
GET-BROWSE-COLUMN	–	–
GET-BUFFER-HANDLE	✓	–
GETBYTE	✓	–
GET-BYTE	✓	–
GET-CALLBACK-PROC-CONTEXT	–	–
GET-CALLBACK-PROC-NAME	–	–
GET-CGI-LIST	–	–
GET-CGI-LONG-VALUE	–	–
GET-CGI-VALUE	–	–
GET-CLASS	–	–
GET-CODEPAGE	✓	–
GET-CODEPAGES	✓	–
GET-COLLATIONS	✓	–
GET-CONFIG-VALUE	–	–
GET-CURRENT	–	–
GET-DOUBLE	–	–
GET-DROPPED-FILE	–	–
GET-DYNAMIC	–	–
GET-ERROR-COLUMN	✓	–
GET-ERROR-ROW	✓	–
GET-FILE	–	–

Keyword	Rsrv	Minimum abbreviation
GET-FILE-NAME	✓	–
GET-FILE-OFFSET	✓	GET-FILE-OFFSE
GET-FIRST	–	–
GET-FLOAT	–	–
GET-GREEN-VALUE	–	GET-GREEN
GET-INDEX-BY-NAMESPACE-NAME	–	–
GET-INDEX-BY-QNAME	–	–
GET-INT64	–	–
GET-ITERATION	–	–
GET-KEY-VALUE	✓	GET-KEY-VAL
GET-LAST	–	–
GET-LOCALNAME-BY-INDEX	–	–
GET-LONG	–	–
GET-MESSAGE	–	–
GET-NEXT	–	–
GET-NUMBER	–	–
GET-POINTER-VALUE	–	–
GET-PREV	–	–
GET-PRINTERS	–	–
GET-PROPERTY	–	–
GET-QNAME-BY-INDEX	–	–
GET-RED-VALUE	–	GET-RED
GET-REPOSITIONED-ROW	–	–
GET-RGB-VALUE	–	–
GET-SELECTED-WIDGET	–	GET-SELECTED

Keyword	Rsrv	Minimum abbreviation
GET-SHORT	–	–
GET-SIGNATURE	–	–
GET-SIZE	–	–
GET-STRING	–	–
GET-TAB-ITEM	–	–
GET-TEXT-HEIGHT-CHARS	–	GET-TEXT-HEIGHT
GET-TEXT-HEIGHT-PIXELS	–	GET-TEXT-HEIGHT-P
GET-TEXT-WIDTH-CHARS	–	GET-TEXT-WIDTH
GET-TEXT-WIDTH-PIXELS	–	GET-TEXT-WIDTH-P
GET-TYPE-BY-INDEX	–	–
GET-TYPE-BY-NAMESPACE-NAME	–	–
GET-TYPE-BY-QNAME	–	–
GET-UNSIGNED-LONG	–	–
GET-UNSIGNED-SHORT	–	–
GET-URI-BY-INDEX	–	–
GET-VALUE-BY-INDEX	–	–
GET-VALUE-BY-NAMESPACE-NAME	–	–
GET-VALUE-BY-QNAME	–	–
GET-WAIT-STATE	–	–
GLOBAL	✓	–
GO-ON	✓	–
GO-PENDING	✓	GO-PEND
GRANT	✓	–
GRAPHIC-EDGE	✓	GRAPHIC-E
GRID-FACTOR-HORIZONTAL	–	GRID-FACTOR-H

Keyword	Rsrv	Minimum abbreviation
GRID-FACTOR-VERTICAL	–	GRID-FACTOR-V
GRID-SNAP	–	–
GRID-UNIT-HEIGHT-CHARS	–	GRID-UNIT-HEIGHT
GRID-UNIT-HEIGHT-PIXELS	–	GRID-UNIT-HEIGHT-P
GRID-UNIT-WIDTH-CHARS	–	GRID-UNIT-WIDTH
GRID-UNIT-WIDTH-PIXELS	–	GRID-UNIT-WIDTH-P
GRID-VISIBLE	–	–
GROUP	✓	–
GT	–	–
GUID	–	–
HANDLE	–	–
HANDLER	–	–
HAS-RECORDS	–	–
HAVING	✓	–
HEADER	✓	–
HEIGHT-CHARS	–	HEIGHT
HEIGHT-PIXELS	–	HEIGHT-P
HELP	✓	–
HEX-DECODE	–	–
HEX-ENCODE	–	–
HIDDEN	–	–
HIDE	✓	–
HORIZONTAL	–	HORI
HOST-BYTE-ORDER	✓	–
HTML-CHARSET	–	–

Keyword	Rsrv	Minimum abbreviation
HTML-END-OF-LINE	–	–
HTML-END-OF-PAGE	–	–
HTML-FRAME-BEGIN	–	–
HTML-FRAME-END	–	–
HTML-HEADER-BEGIN	–	–
HTML-HEADER-END	–	–
HTML-TITLE-BEGIN	–	–
HTML-TITLE-END	–	–
HWND	–	–
ICON	–	–
IF	✓	–
IMAGE	–	–
IMAGE-DOWN	–	–
IMAGE-INSENSITIVE	–	–
IMAGE-SIZE	–	–
IMAGE-SIZE-CHARS	–	IMAGE-SIZE-C
IMAGE-SIZE-PIXELS	–	IMAGE-SIZE-P
IMAGE-UP	–	–
IMMEDIATE-DISPLAY	–	–
IMPLEMENTS	–	–
IMPORT	✓	–
IMPORT-PRINCIPAL	–	–
IN	✓	–
INCREMENT-EXCLUSIVE-ID	–	–
INDEX	✓	–

Keyword	Rsrv	Minimum abbreviation
INDEXED-REPOSITION	–	–
INDEX-HINT	–	–
INDEX-INFORMATION	–	–
INDICATOR	✓	–
INFORMATION	–	INFO
IN-HANDLE	–	–
INHERIT-BGCOLOR	–	INHERIT-BGC
INHERIT-FGCOLOR	–	INHERIT-FGC
INHERITS	–	–
INITIAL	–	INIT
INITIAL-DIR	–	–
INITIAL-FILTER	–	–
INITIALIZE-DOCUMENT-TYPE	–	–
INITIATE	–	–
INNER-CHARS	–	–
INNER-LINES	–	–
INPUT	✓	–
INPUT-OUTPUT	✓	INPUT-O
INPUT-VALUE	–	–
INSERT	✓	–
INSERT-ATTRIBUTE	–	–
INSERT-BACKTAB	–	INSERT-B
INSERT-FILE	–	–
INSERT-ROW	–	–
INSERT-STRING	–	–

Keyword	Rsrv	Minimum abbreviation
INSERT-TAB	–	INSERT-T
INT	–	–
INT64	–	–
INTEGER	–	INT
INTERFACE	–	–
INTERNAL-ENTRIES	–	–
INTO	✓	–
INVOKE	–	–
IS	✓	–
IS-ATTR-SPACE	✓	IS-ATTR
IS-CLASS	–	IS-CLAS
IS-JSON	–	–
IS-LEAD-BYTE	✓	IS-ATTR
IS-OPEN	–	–
IS-PARAMETER-SET	–	–
IS-PARTITIONED	–	IS-PARTITIONE
IS-ROW-SELECTED	–	–
IS-SELECTED	–	–
IS-XML	–	–
ITEM	–	–
ITEMS-PER-ROW	–	–
JOIN	✓	–
JOIN-BY-SQLDB	–	–
KBLABEL	✓	–
KEEP-CONNECTION-OPEN	–	–

Keyword	Rsrv	Minimum abbreviation
KEEP-FRAME-Z-ORDER	–	KEEP-FRAME-Z
KEEP-MESSAGES	–	–
KEEP-SECURITY-CACHE	–	–
KEEP-TAB-ORDER	–	–
KEY	–	–
KEYCACHE-JOIN	–	–
KEYCODE	✓	–
KEY-CODE	✓	–
KEYFUNCTION	✓	KEYFUNC
KEY-FUNCTION	✓	KEY-FUNC
KEYLABEL	✓	–
KEY-LABEL	✓	–
KEYS	✓	–
KEYWORD	✓	–
KEYWORD-ALL	–	–
LABEL	✓	–
LABEL-BGCOLOR	–	LABEL-BGC
LABEL-DCOLOR	–	LABEL-DC
LABEL-FGCOLOR	–	LABEL-FGC
LABEL-FONT	–	–
LABEL-PFCOLOR	–	LABEL-PFC
LABELS	–	–
LABELS-HAVE-COLONS	–	–
LANDSCAPE	–	–
LANGUAGES	–	LANGUAGE

Keyword	Rsrv	Minimum abbreviation
LARGE	–	–
LARGE-TO-SMALL	–	–
LAST	✓	–
LAST-ASYNCH-REQUEST	–	–
LAST-BATCH	–	–
LAST-CHILD	–	–
LAST-EVENT	✓	LAST-EVEN
LAST-FORM	–	–
LASTKEY	✓	–
LAST-KEY	✓	–
LAST-OBJECT	–	–
LAST-OF	✓	–
LAST-PROCEDURE	–	LAST-PROCE
LAST-SERVER	–	–
LAST-TAB-ITEM	–	LAST-TAB-I
LC	–	–
LDBNAME	✓	–
LE	–	–
LEAVE	✓	–
LEFT-ALIGNED	–	LEFT-ALIGN
LEFT-TRIM	–	–
LENGTH	–	–
LIBRARY	✓	–
LIKE	✓	–
LIKE-SEQUENTIAL	✓	–

Keyword	Rsrv	Minimum abbreviation
LINE	–	–
LINE-COUNTER	✓	LINE-COUNT
LIST-EVENTS	–	–
LISTING	✓	LISTI
LIST-ITEM-PAIRS	–	–
LIST-ITEMS	–	–
LIST-PROPERTY-NAMES	–	–
LIST-QUERY-ATTRS	–	–
LIST-SET-ATTRS	–	–
LIST-WIDGETS	–	–
LITERAL-QUESTION	–	–
LITTLE-ENDIAN	✓	–
LOAD	–	–
LOAD-DOMAINS	–	–
LOAD-ICON	–	–
LOAD-IMAGE	–	–
LOAD-IMAGE-DOWN	–	–
LOAD-IMAGE-INSENSITIVE	–	–
LOAD-IMAGE-UP	–	–
LOAD-MOUSE-POINTER	–	LOAD-MOUSE-P
LOAD-PICTURE	–	–
LOAD-SMALL-ICON	–	–
LOCAL-NAME	–	–
LOCAL-VERSION-INFO	–	–
LOCATOR-COLUMN-NUMBER	–	–

Keyword	Rsrv	Minimum abbreviation
LOCATOR-LINE-NUMBER	–	–
LOCATOR-PUBLIC-ID	–	–
LOCATOR-SYSTEM-ID	–	–
LOCATOR-TYPE	–	–
LOCKED	✓	–
LOCK-REGISTRATION	–	–
LOG	–	–
LOG-AUDIT-EVENT	–	–
LOGICAL	–	–
LOGIN-EXPIRATION-TIMESTAMP	–	–
LOGIN-HOST	–	–
LOGIN-STATE	–	–
LOG-MANAGER	✓	–
LOGOUT	–	–
LOOKAHEAD	–	–
LOOKUP	✓	–
LT	–	–
MACHINE-CLASS	✓	–
MANDATORY	–	–
MANUAL-HIGHLIGHT	–	–
MAP	✓	–
MARGIN-EXTRA	–	–
MARGIN-HEIGHT-CHARS	–	MARGIN-HEIGHT
MARGIN-HEIGHT-PIXELS	–	MARGIN-HEIGHT-P
MARGIN-WIDTH-CHARS	–	MARGIN-WIDTH

Keyword	Rsrv	Minimum abbreviation
MARGIN-WIDTH-PIXELS	–	MARGIN-WIDTH-P
MARK-NEW	–	–
MARK-ROW-STATE	–	–
MATCHES	–	–
MAX	–	–
MAX-BUTTON	–	–
MAX-CHARS	–	–
MAX-DATA-GUESS	–	–
MAX-HEIGHT	–	–
MAX-HEIGHT-CHARS	–	MAX-HEIGHT-C
MAX-HEIGHT-PIXELS	–	MAX-HEIGHT-P
MAXIMIZE	–	–
MAXIMUM	–	MAX
MAXIMUM-LEVEL	–	–
MAX-ROWS	–	–
MAX-SIZE	–	–
MAX-VALUE	–	MAX-VAL
MAX-WIDTH	–	–
MAX-WIDTH-CHARS	–	MAX-WIDTH
MAX-WIDTH-PIXELS	–	MAX-WIDTH-P
MD5-DIGEST	–	–
MEMBER	✓	–
MEMPTR-TO-NODE-VALUE	–	–
MENU	–	–
MENUBAR	–	–
MENU-BAR	–	–

Keyword	Rsrv	Minimum abbreviation
MENU-ITEM	–	–
MENU-KEY	–	MENU-K
MENU-MOUSE	–	MENU-M
MERGE-BY-FIELD	–	–
MESSAGE	✓	–
MESSAGE-AREA	–	–
MESSAGE-AREA-FONT	–	–
MESSAGE-LINES	✓	–
METHOD	–	–
MIN	–	–
MIN-BUTTON	–	–
MIN-COLUMN-WIDTH-CHARS	–	MIN-COLUMN-WIDTH-C
MIN-COLUMN-WIDTH-PIXELS	–	MIN-COLUMN-WIDTH-P
MIN-HEIGHT-CHARS	–	MIN-HEIGHT
MIN-HEIGHT-PIXELS	–	MIN-HEIGHT-P
MINIMUM	–	MIN
MIN-SIZE	–	–
MIN-VALUE	–	MIN-VAL
MIN-WIDTH-CHARS	–	MIN-WIDTH
MIN-WIDTH-PIXELS	–	MIN-WIDTH-P
MODIFIED	–	–
MODULO	–	MOD
MONTH	–	–
MOUSE	✓	–
MOUSE-POINTER	–	MOUSE-P

Keyword	Rsrv	Minimum abbreviation
MOVABLE	–	–
MOVE-AFTER-TAB-ITEM	–	MOVE-AFTER
MOVE-BEFORE-TAB-ITEM	–	MOVE-BEFOR
MOVE-COLUMN	–	MOVE-COL
MOVE-TO-BOTTOM	–	MOVE-TO-B
MOVE-TO-EOF	–	–
MOVE-TO-TOP	–	MOVE-TO-T
MPE	✓	–
MTIME	–	–
MULTI-COMPILE	–	–
MULTIPLE	–	–
MULTIPLE-KEY	–	–
MULTITASKING-INTERVAL	–	–
MUST-EXIST	–	–
NAME	–	–
NAMESPACE-PREFIX	–	–
NAMESPACE-URI	–	–
NATIVE	–	–
NE	–	–
NEEDS-APPSERVER-PROMPT	–	–
NEEDS-PROMPT	–	–
NEW	✓	–
NEW-INSTANCE	–	–
NEW-ROW	–	–
NEXT	✓	–

Keyword	Rsrv	Minimum abbreviation
NEXT-COLUMN	–	–
NEXT-PROMPT	✓	–
NEXT-ROWID	–	–
NEXT-SIBLING	–	–
NEXT-TAB-ITEM	–	NEXT-TAB-I
NEXT-VALUE	–	–
NO	✓	–
NO-APPLY	–	–
NO-ARRAY-MESSAGE	–	–
NO-ASSIGN	–	–
NO-ATTR-LIST	✓	NO-ATTR
NO-ATTR-SPACE	✓	NO-ATTR
NO-AUTO-VALIDATE	–	–
NO-BIND-WHERE	–	–
NO-BOX	–	–
NO-CONSOLE	–	–
NO-CONVERT	–	–
NO-CONVERT-3D-COLORS	–	–
NO-CURRENT-VALUE	–	–
NO-DEBUG	–	–
NODE-VALUE-TO-MEMPTR	–	–
NO-DRAG	–	–
NO-ECHO	–	–
NO-EMPTY-SPACE	–	–
NO-ERROR	✓	–

Keyword	Rsrv	Minimum abbreviation
NO-FILL	✓	NO-F
NO-FOCUS	✓	–
NO-HELP	–	–
NO-HIDE	✓	–
NO-INDEX-HINT	–	–
NO-INHERIT-BGCOLOR	–	NO-INHERIT-BGC
NO-INHERIT-FGCOLOR	–	NO-INHERIT-FGC
NO-JOIN-BY-SQLDB	–	–
NO-KEYCACHE-JOIN	–	–
NO-LABELS	✓	NO-LABEL
NO-LOBS	✓	–
NO-LOCK	✓	–
NO-LOOKAHEAD	–	–
NO-MAP	✓	–
NO-MESSAGE	✓	NO-MES
NONAMESPACE-SCHEMA-LOCATION	–	–
NONE	–	–
NON-SERIALIZABLE	–	–
NO-PAUSE	✓	–
NO-PREFETCH	✓	NO-PREFE
NORMALIZE	–	–
NO-ROW-MARKERS	–	–
NO-SCROLLBAR-VERTICAL	–	–
NO-SEPARATE-CONNECTION	–	–
NO-SEPARATORS	–	–

Keyword	Rsrv	Minimum abbreviation
NOT	✓	–
NO-TAB-STOP	–	–
NOT-ACTIVE	–	–
NO-UNDERLINE	–	NO-UND
NO-UNDO	✓	–
NO-VALIDATE	✓	NO-VAL
NOW	✓	–
NO-WAIT	✓	–
NO-WORD-WRAP	–	–
NULL	✓	–
NUM-ALIASES	✓	NUM-ALI
NUM-BUFFERS	–	–
NUM-BUTTONS	–	NUM-BUT
NUM-COLUMNS	–	NUM-COL
NUM-COPIES	–	–
NUM-DBS	✓	–
NUM-DROPPED-FILES	–	–
NUM-ENTRIES	✓	–
NUMERIC	–	–
NUMERIC-FORMAT	–	NUMERIC-F
NUM-FIELDS	–	–
NUM-FORMATS	–	–
NUM-ITEMS	–	–
NUM-ITERATIONS	–	–
NUM-LINES	–	–

Keyword	Rsrv	Minimum abbreviation
NUM-LOCKED-COLUMNS	–	NUM-LOCKED-COL
NUM-MESSAGES	–	–
NUM-PARAMETERS	–	–
NUM-REFERENCES	–	–
NUM-REPLACED	–	–
NUM-RESULTS	–	–
NUM-SELECTED-ROWS	–	–
NUM-SELECTED-WIDGETS	–	NUM-SELECTED
NUM-TABS	–	–
NUM-TO-RETAIN	–	–
NUM-VISIBLE-COLUMNS	–	–
OCTET-LENGTH	–	–
OF	✓	–
OFF	✓	–
OK	–	–
OK-CANCEL	–	–
OLD	✓	–
ON	✓	–
ON-FRAME-BORDER	–	ON-FRAME
OPEN	✓	–
OPSYS	✓	–
OPTION	✓	–
OPTIONS	–	–
OPTIONS-FILE	–	–
OR	✓	–

Keyword	Rsrv	Minimum abbreviation
ORDERED-JOIN	–	–
ORDINAL	–	–
OS-APPEND	✓	–
OS-COMMAND	✓	–
OS-COPY	✓	–
OS-CREATE-DIR	✓	–
OS-DELETE	✓	–
OS-DIR	✓	–
OS-DRIVES	–	OS-DRIVE
OS-ERROR	–	–
OS-GETENV	–	–
OS-RENAME	✓	–
OTHERWISE	✓	–
OUTPUT	✓	–
OVERLAY	✓	–
OVERRIDE	–	–
OWNER	–	–
PACKAGE-PRIVATE	–	–
PACKAGE-PROTECTED	–	–
PAGE	✓	–
PAGE-BOTTOM	✓	PAGE-BOT
PAGED	–	–
PAGE-NUMBER	✓	PAGE-NUM
PAGE-SIZE	–	–

Keyword	Rsrv	Minimum abbreviation
PAGE-TOP	✓	–
PAGE-WIDTH	–	PAGE-WID
PARAMETER	✓	PARAM
PARENT	–	–
PARSE-STATUS	–	–
PARTIAL-KEY	–	–
PASCAL	–	–
PASSWORD-FIELD	✓	–
PATHNAME	–	–
PAUSE	✓	–
PBE-HASH-ALGORITHM	–	PBE-HASH-ALG
PBE-KEY-ROUNDS	–	–
PDBNAME	✓	–
PERSISTENT	✓	PERSIST
PERSISTENT-CACHE-DISABLED	–	–
PFCOLOR	–	PFC
PIXELS	✓	–
PIXELS-PER-COLUMN	–	PIXELS-PER-COL
PIXELS-PER-ROW	–	–
POPUP-MENU	–	POPUP-M
POPUP-ONLY	–	POPUP-O
PORTRAIT	–	–
POSITION	–	–
PRECISION	–	–
PREFER-DATASET	–	–

Keyword	Rsrv	Minimum abbreviation
PREPARED	–	–
PREPARE-STRING	–	–
PREPROCESS	✓	PREPROC
PRESELECT	–	PRESEL
PREV	–	–
PREV-COLUMN	–	–
PREV-SIBLING	–	–
PREV-TAB-ITEM	–	PREV-TAB-I
PRIMARY	–	–
PRINTER	–	–
PRINTER-CONTROL-HANDLE	–	–
PRINTER-HDC	–	–
PRINTER-NAME	–	–
PRINTER-PORT	–	–
PRINTER-SETUP	–	–
PRIVATE	–	–
PRIVATE-DATA	–	PRIVATE-D
PRIVILEGES	✓	–
PROCEDURE	–	PROCE
PROCEDURE-CALL-TYPE	✓	–
PROCEDURE-TYPE	–	–
PROCESS	✓	–
PROC-HANDLE	✓	PROC-HA
PROC-STATUS	✓	PROC-ST
proc-text	–	–

Keyword	Rsrv	Minimum abbreviation
proc-text-buffer	–	–
PROFILER	✓	–
PROGRAM-NAME	✓	–
PROGRESS	✓	–
PROGRESS-SOURCE	–	PROGRESS-S
PROMPT	–	–
PROMPT-FOR	✓	PROMPT-F
PROMSGS	✓	–
PROPATH	✓	–
PROPERTY	–	–
PROTECTED	–	–
PROVERSION	✓	PROVERS
PROXY	–	–
PROXY-PASSWORD	–	–
PROXY-USERID	–	–
PUBLIC	–	–
PUBLIC-ID	–	–
PUBLISH	–	–
PUBLISHED-EVENTS	–	–
PUT	✓	–
PUTBYTE	✓	–
PUT-BYTE	✓	–
PUT-DOUBLE	–	–
PUT-FLOAT	–	–
PUT-INT64	–	–

Keyword	Rsrv	Minimum abbreviation
PUT-KEY-VALUE	✓	PUT-KEY-VAL
PUT-LONG	–	–
PUT-SHORT	–	–
PUT-STRING	–	–
PUT-UNSIGNED-LONG	–	–
QUERY	✓	–
QUERY-CLOSE	✓	–
QUERY-OFF-END	✓	–
QUERY-OPEN	–	–
QUERY-PREPARE	–	–
QUERY-TUNING	✓	–
QUESTION	–	–
QUIT	✓	–
QUOTER	–	–
RADIO-BUTTONS	–	–
RADIO-SET	–	–
RANDOM	–	–
RAW	–	–
RAW-TRANSFER	–	–
RCODE-INFORMATION	✓	RCODE-INFO
READ-AVAILABLE	✓	–
READ-EXACT-NUM	✓	–
READ-FILE	–	–
READ-JSON	–	–
READ-ONLY	–	–

Keyword	Rsrv	Minimum abbreviation
READ-XML	–	–
READKEY	✓	–
REAL	–	–
RECID	✓	–
RECORD-LENGTH	–	–
RECTANGLE	✓	RECT
RECURSIVE	–	–
REFERENCE-ONLY	–	–
REFRESH	–	–
REFRESHABLE	–	–
REFRESH-AUDIT-POLICY	–	–
REGISTER-DOMAIN	–	–
RELEASE	✓	–
REMOTE	–	–
REMOVE-EVENTS-PROCEDURE	–	–
REMOVE-SUPER-PROCEDURE	–	–
REPEAT	✓	–
REPLACE	–	–
REPLACE-SELECTION-TEXT	–	–
REPOSITION	✓	–
REPOSITION-BACKWARD	✓	–
REPOSITION-FORWARD	✓	–
REPOSITION-MODE	–	–
REPOSITION-TO-ROW	✓	–
REPOSITION-TO-ROWID	✓	–

Keyword	Rsrv	Minimum abbreviation
REQUEST	–	–
REQUEST-INFO	–	–
RESET	–	–
RESIZABLE	–	RESIZA
RESIZE	–	–
RESPONSE-INFO	–	–
RESTART-ROW	–	–
RESTART-ROWID	–	–
RETAIN	✓	–
RETAIN-SHAPE	–	–
RETRY	✓	–
RETRY-CANCEL	–	–
RETURN	✓	–
RETURN-INSERTED	–	RETURN-INS
RETURNS	✓	–
RETURN-TO-START-DIR	–	RETURN-TO-START-DI
RETURN-VALUE	–	RETURN-VAL
RETURN-VALUE-DATA-TYPE	–	–
REVERSE-FROM	–	–
REVERT	✓	–
REVOKE	✓	–
RGB-VALUE	–	–
RIGHT-ALIGNED	–	RETURN-ALIGN
RIGHT-TRIM	–	–
R-INDEX	✓	–

Keyword	Rsrv	Minimum abbreviation
ROLES	–	–
ROUND	–	–
ROUTINE-LEVEL	–	–
ROW	–	–
ROW-HEIGHT-CHARS	–	HEIGHT
ROW-HEIGHT-PIXELS	–	HEIGHT-P
ROWID	✓	–
ROW-MARKERS	–	–
ROW-OF	–	–
ROW-RESIZABLE	–	–
RULE	–	–
RUN	✓	–
RUN-PROCEDURE	–	–
SAVE	✓	–
SAVE CACHE	–	–
SAVE-AS	–	–
SAVE-FILE	–	–
SAX-COMPLETE	✓	SAX-COMPLE
SAX-PARSE	–	–
SAX-PARSE-FIRST	–	–
SAX-PARSE-NEXT	–	–
SAX-PARSER-ERROR	✓	–
SAX-RUNNING	✓	–
SAX-UNINITIALIZED	✓	–
SAX-WRITE-BEGIN	✓	–

Keyword	Rsrv	Minimum abbreviation
SAX-WRITE-COMPLETE	✓	–
SAX-WRITE-CONTENT	✓	–
SAX-WRITE-ELEMENT	✓	–
SAX-WRITE-ERROR	✓	–
SAX-WRITE-IDLE	✓	–
SAX-WRITER	–	–
SAX-WRITE-TAG	✓	–
SCHEMA	✓	–
SCHEMA-LOCATION	–	–
SCHEMA-MARSHAL	–	–
SCHEMA-PATH	–	–
SCREEN	✓	–
SCREEN-IO	✓	–
SCREEN-LINES	✓	–
SCREEN-VALUE	–	SCREEN-VAL
SCROLL	✓	–
SCROLLABLE	–	–
SCROLLBAR-HORIZONTAL	–	SCROLLBAR-H
SCROLL-BARS	–	–
SCROLLBAR-VERTICAL	–	SCROLLBAR-V
SCROLL-DELTA	–	–
SCROLLED-ROW-POSITION	–	SCROLLED-ROW-POS
SCROLLING	–	–
SCROLL-OFFSET	–	–

Keyword	Rsrv	Minimum abbreviation
SCROLL-TO-CURRENT-ROW	–	–
SCROLL-TO-ITEM	–	SCROLL-TO-I
SCROLL-TO-SELECTED-ROW	–	–
SDBNAME	✓	–
SEAL	–	–
SEAL-TIMESTAMP	–	–
SEARCH	✓	–
SEARCH-SELF	✓	–
SEARCH-TARGER	✓	–
SECTION	–	–
SECURITY-POLICY	✓	–
SEEK	✓	–
SELECT	✓	–
SELECTABLE	–	–
SELECT-ALL	–	–
SELECTED	–	–
SELECT-FOCUSED-ROW	–	–
SELECTION	–	–
SELECTION-END	–	–
SELECTION-LIST	–	–
SELECTION-START	–	–
SELECTION-TEXT	–	–
SELECT-NEXT-ROW	–	–
SELECT-PREV-ROW	–	–
SELECT-ROW	–	–

Keyword	Rsrv	Minimum abbreviation
SELF	✓	–
SEND	–	–
send-sql-statement	–	send-sql
SENSITIVE	–	–
SEPARATE-CONNECTION	–	–
SEPARATOR-FGCOLOR	–	–
SEPARATORS	–	–
SERIALIZABLE	–	–
SERIALIZE-HIDDEN	–	–
SERIALIZE-NAME	–	–
SERVER	–	–
SERVER-CONNECTION-BOUND	–	–
SERVER-CONNECTION-BOUND-REQUEST	–	–
SERVER-CONNECTION-CONTEXT	–	–
SERVER-CONNECTION-ID	–	–
SERVER-OPERATING-MODE	–	–
SESSION	✓	–
SESSION-ID	–	–
SET	✓	–
SET-APPL-CONTEXT	–	–
SET-ATTR-CALL-TYPE	✓	–
SET-ATTRIBUTE-NODE	–	–
SET-BLUE-VALUE	–	SET-BLUE
SET-BREAK	–	–
SET-BUFFERS	–	–

Keyword	Rsrv	Minimum abbreviation
SET-CALLBACK	–	–
SET-CLIENT	–	–
SET-COMMIT	–	–
SET-CONTENTS	–	–
SET-CURRENT-VALUE	–	–
SET-DB-CLIENT	–	–
SET-DYNAMIC	–	–
SET-EVENT-MANAGER-OPTION	–	–
SET-GREEN-VALUE	–	SET-GREEN
SET-INPUT-SOURCE	–	–
SET-OPTION	–	–
SET-OUTPUT-DESTINATION	–	–
SET-PARAMETER	–	–
SET-POINTER-VALUE	–	–
SET-PROPERTY	–	–
SET-RED-VALUE	–	SET-RED
SET-REPOSITIONED-ROW	–	–
SET-RGB-VALUE	–	–
SET-ROLLBACK	–	–
SET-SELECTION	–	–
SET-SIZE	–	–
SET-SORT-ARROW	–	–
SETUSERID	✓	SETUSER
SET-WAIT-STATE	–	–
SHA1-DIGEST	–	–

Keyword	Rsrv	Minimum abbreviation
SHARED	✓	–
SHARE-LOCK	✓	SHARE
SHOW-IN-TASKBAR	–	–
SHOW-STATS	✓	SHOW-STAT
SIDE-LABEL-HANDLE	–	SIDE-LABEL-H
SIDE-LABELS	–	SIDE-LAB
SIGNATURE	–	–
SIGNATURE-VALUE	–	–
SILENT	–	–
SIMPLE	–	–
SINGLE	–	–
SINGLE-RUN	✓	–
SINGLETON	–	–
SIZE	–	–
SIZE-CHARS	–	SIZE-C
SIZE-PIXELS	–	SIZE-P
SKIP	✓	–
SKIP-DELETED-RECORD	✓	–
SLIDER	–	–
SMALL-ICON	–	–
SMALLINT	–	–
SMALL-TITLE	–	–
SOME	✓	–
SORT	–	–
SORT-ASCENDING	–	–

Keyword	Rsrv	Minimum abbreviation
SORT-NUMBER	–	–
SOURCE	–	–
SOURCE-PROCEDURE	–	–
SPACE	✓	–
SQL	–	–
SQRT	–	–
SSL-SERVER-NAME	–	–
STANDALONE	–	–
START	–	–
START-DOCUMENT	–	–
START-ELEMENT	–	–
START-MOVE	–	–
START-RESIZE	–	–
START-ROW-RESIZE	–	–
STATE-DETAIL	–	–
STATIC	–	–
STATUS	✓	–
STATUS-AREA	–	–
STATUS-AREA-FONT	–	–
STDCALL	–	–
STOP	–	–
STOP-AFTER	–	–
STOP-OBJECT	–	–
STOP-PARSING	–	–
STOPPED	–	STOPPE

Keyword	Rsrv	Minimum abbreviation
STORED-PROCEDURE	–	STORED-PROC
STREAM	✓	–
STREAM-HANDLE	✓	–
STREAM-IO	✓	–
STRETCH-TO-FIT	–	–
STRICT	–	–
STRICT-ENTITY-RESOLUTION	–	–
STRING	–	–
STRING-VALUE	–	–
STRING-XREF	–	–
SUB-AVERAGE	–	SUB-AVE
SUB-COUNT	–	–
SUB-MAXIMUM	–	SUM-MAX
SUB-MENU	–	SUB-
SUB-MINIMUM	–	SUB-MIN
SUBSCRIBE	–	–
SUBSTITUTE	–	SUBST
SUBSTRING	–	SUBSTR
SUB-TOTAL	–	–
SUBTYPE	–	–
SUM	–	–
SUPER	–	–
SUPER-PROCEDURES	–	–
SUPPRESS-NAMESPACE-PROCESSING	–	–
SUPPRESS-WARNINGS	–	SUPPRESS-W

Keyword	Rsrv	Minimum abbreviation
SYMMETRIC-ENCRYPTION-ALGORITHM	–	–
SYMMETRIC-ENCRYPTION-IV	–	–
SYMMETRIC-ENCRYPTION-KEY	–	–
SYMMETRIC-SUPPORT	–	–
SYSTEM-ALERT-BOXES	–	SYSTEM-ALERT
SYSTEM-DIALOG	✓	–
SYSTEM-HELP	–	–
SYSTEM-ID	–	–
TABLE	✓	–
TABLE-HANDLE	–	–
TABLE-NUMBER	✓	–
TABLE-SCAN	–	–
TAB-POSITION	–	–
TAB-STOP	–	–
TARGET	–	–
TARGET-PROCEDURE	–	–
TEMP-DIRECTORY	–	TEMP-DIR
TEMP-TABLE	–	–
TEMP-TABLE-PREPARE	–	–
TERM	✓	–
TERMINAL	✓	TERM
TERMINATE	–	–
TEXT	✓	–
TEXT-CURSOR	✓	–
TEXT-SEG-GROW	–	–

Keyword	Rsrv	Minimum abbreviation
TEXT-SELECTED	–	–
THEN	✓	–
THIS-OBJECT	✓	–
THIS-PROCEDURE	✓	–
THREAD-SAFE	✓	–
THREE-D	–	–
THROW	–	–
THROUGH	–	–
THRU	–	–
TIC-MARKS	–	–
TIME	✓	–
TIME-SOURCE	–	–
TITLE	✓	–
TITLE-BGCOLOR	–	TITLE-BGC
TITLE-DCOLOR	–	TITLE-DC
TITLE-FGCOLOR	–	TITLE-FGC
TITLE-FONT	–	TITLE-FO
TO	✓	–
TODAY	–	–
TOGGLE-BOX	–	–
TOOLTIP	–	–
TOOLTIPS	–	–
TOPIC	–	–
TOP-NAV-QUERY	–	–
TOP-ONLY	✓	–

Keyword	Rsrv	Minimum abbreviation
TO-ROWID	✓	–
TOTAL	–	–
TRAILING	–	–
TRANS	✓	–
TRANSACTION	✓	–
TRANSACTION-MODE	–	–
TRANS-INIT-PROCEDURE	–	–
TRANSPARENT	–	–
TRIGGER	✓	–
TRIGGERS	✓	–
TRIM	✓	–
TRUE	✓	–
TRUNCATE	–	TRUNC
TYPE	–	–
TYPE-OF	–	–
UNBOX	–	–
UNBUFFERED	–	UNBUFF
UNDERLINE	✓	UNDERL
UNDO	✓	–
UNFORMATTED	✓	UNFORM
UNION	✓	–
UNIQUE	✓	–
UNIQUE-ID	–	–
UNIQUE-MATCH	–	–

Keyword	Rsrv	Minimum abbreviation
UNIX	✓	–
UNLESS-HIDDEN	–	–
UNLOAD	–	–
UNSIGNED-LONG	–	–
UNSUBSCRIBE	–	–
UP	✓	–
UPDATE	✓	–
UPDATE-ATTRIBUTE	–	–
URL	–	–
URL-DECODE	–	–
URL-ENCODE	–	–
URL-PASSWORD	–	–
URL-USERID	–	–
USE	–	–
USE-DICT-EXPS	–	–
USE-FILENAME	–	–
USE-INDEX	✓	–
USER	–	–
USE-REVVIDEO	–	–
USERID	✓	–
USER-ID	–	–
USE-TEXT	–	–
USE-UNDERLINE	–	–
USE-WIDGET-POOL	–	–
USING	✓	–

Keyword	Rsrv	Minimum abbreviation
V6DISPLAY	–	–
V6FRAME	✓	–
VALIDATE	–	–
VALIDATE-DOMAIN-ACCESS-CODE	–	–
VALIDATE-EXPRESSION	–	–
VALIDATE-MESSAGE	–	–
VALIDATE-SEAL	–	–
VALIDATION-ENABLED	–	–
VALID-EVENT	–	–
VALID-HANDLE	–	–
VALID-OBJECT	–	–
VALUE	✓	–
VALUE-CHANGED	✓	–
VALUES	✓	–
VAR	–	–
VARIABLE	–	VAR
VERBOSE	–	–
VERSION	–	–
VERTICAL	–	VERT
VIEW	✓	–
VIEW-AS	✓	–
VIEW-FIRST-COLUMN-ON-REOPEN	–	–
VIRTUAL-HEIGHT-CHARS	–	VIRTUAL-HEIGHT
VIRTUAL-HEIGHT-PIXELS	–	VIRTUAL-HEIGHT-P
VIRTUAL-WIDTH-CHARS	–	VIRTUAL-WIDTH

Keyword	Rsrv	Minimum abbreviation
VIRTUAL-WIDTH-PIXELS	–	VIRTUAL-WIDTH-P
VISIBLE	–	–
VOID	–	–
WAIT	–	–
WAIT-FOR	✓	–
WARNING	–	–
WEB-CONTEXT	–	–
WEEKDAY	–	–
WHEN	✓	–
WHERE	✓	–
WHILE	✓	–
WIDGET	–	–
WIDGET-ENTER	–	WIDGET-E
WIDGET-ID	–	–
WIDGET-LEAVE	–	WIDGET-L
WIDGET-POOL	–	–
WIDTH	–	–
WIDTH-CHARS	–	WIDTH
WIDTH-PIXELS	–	WIDTH-P
WINDOW	✓	–
WINDOW-MAXIMIZED	✓	WINDOW-MAXIM
WINDOW-MINIMIZED	✓	WINDOW-MINIM
WINDOW-NAME	–	–
WINDOW-NORMAL	✓	–
WINDOW-STATE	–	WINDOW-STA

Keyword	Rsrv	Minimum abbreviation
WINDOW-SYSTEM	–	–
WITH	✓	–
WORD-INDEX	–	–
WORD-WRAP	–	–
WORK-AREA-HEIGHT-PIXELS	–	–
WORK-AREA-WIDTH-PIXELS	–	–
WORK-AREA-X	–	–
WORK-AREA-Y	–	–
WORKFILE	✓	–
WORK-TABLE	✓	WORK-TAB
WRITE	✓	–
WRITE-CDATA	–	–
WRITE-CHARACTERS	–	–
WRITE-COMMENT	–	–
WRITE-DATA-ELEMENT	–	–
WRITE-EMPTY-ELEMENT	–	–
WRITE-ENTITY-REF	–	–
WRITE-EXTERNAL-DTD	–	–
WRITE-FRAGMENT	–	–
WRITE-JSON	–	–
WRITE-MESSAGE	–	–
WRITE-PROCESSING-INSTRUCTION	–	–
WRITE-STATUS	–	–
WRITE-XML	–	–
WRITE-XMLSCHEMA	–	–

Keyword	Rsrv	Minimum abbreviation
X	–	–
XCODE	✓	–
XML-DATA-TYPE	–	–
XML-ENTITY-EXPANSION-LIMIT	–	–
XML-NODE-TYPE	–	–
XML-SCHEMA-PATH	–	–
XML-STRICT-ENTITY-RESOLUTION	–	–
XML-SUPPRESS-NAMESPACE-PROCESSING	–	–
X-OF	–	–
XREF	✓	–
XREF-XML	✓	–
Y	–	–
YEAR	–	–
YEAR-OFFSET	–	–
YES	✓	–
YES-NO	–	–
YES-NO-CANCEL	–	–
Y-OF	–	–

Index

- operator
  - date subtraction *95*
  - date time subtraction *96*
  - unary negative *94*
  - subtraction *94*

-= operator
  - date subtraction *98*
  - date time subtraction *98*
  - subtraction *98*

, (comma)
  - punctuation *63*

; (semicolon)
  - punctuation *63*
  - special character *63*

: :(double colon)
  - punctuation *62*

: (colon)
  - punctuation *61*

? (question mark)
  - special character *63*

. (period)
  - punctuation *62*

' (single quote)
  - special character *66*

" " (character-string literal) *67*

" (double quote)
  - special character *66*

( ) (parentheses)
  - expression precedence *66*

{} (curly braces)
  - argument reference *68*
  - include file reference *70*
  - preprocessor name reference *74*

@ option
  - DEFINE BROWSE statement *366*
  - DISPLAY statement *531*

* operator
  - multiplication *100*

*= operator
  - multiplication *101*

/ (slash)
  - special character *66*

/ operator
  - division *102*

/* */ Multi-line Comment Characters *84*

// Single-line Comment Characters *85*

/= operator
  - division *103*

\ (backslash)
  - special character *64*

&ELSE preprocessor directive *79*

&ELSEIF preprocessor directive *79*

&ENDIF preprocessor directive *79*

&GLOBAL-DEFINE preprocessor directive *78*

&IF preprocessor directive *79*

&MESSAGE preprocessor directive *82*

&SCOPED-DEFINE preprocessor directive *83*

&THEN preprocessor directive *79*

&UNDEFINE preprocessor directive *84*

^ (caret)
  - PROMPT-FOR statement *568*

^ (caret) option
  - IMPORT statement *717*

^ (ignore input) option
  - PROMPT-FOR statement *929*
  - SET statement *1064*
  - UPDATE statement *1157*

+ operator
  - addition *87*
  - concatenation *88*
  - date addition *89*
  - date time addition *90*
  - unary positive *86*

+= operator
  - addition *91*
  - concatenation *91*
  - date addition *91*
  - date time addition *91*

< = Special Character *67*

< > Special Character *67*

< Special Character *67*

= operator
  - assignment statement *122*
  - DYNAMIC-NEW statement *555*
  - NEW statement *826*

= Special Character *67*

> Special Character *67*

>= Special Character *67*

~ (tilde)
  - special character *65*

A

ABL object types *315*

ABL primitive types *315*

ABL-derived .NET class *315*

ABL-extended .NET class *315*

ABSOLUTE function *105*

ABSTRACT option
  - CLASS statement *188*
  - DEFINE EVENT statement *404*
  - DEFINE PROPERTY statement *442*

ABSTRACT option *(continued)*
METHOD statement *808*
ACCELERATOR attribute *1362*
ACCELERATOR option
DEFINE MENU statement *423*
DEFINE SUB-MENU statement *471*
ACCEPT-CHANGES method *1362*
ACCEPT-ROW-CHANGES method *1362*
AccessDenied property *2053*
AccessMode property *2052*
Accessor access mode
DEFINE PROPERTY statement *442*
AccessRevoked property *2053*
AccountDisabled property *2053*
AccountExpired property *2053*
ACCUM function *106*
ACCUM option
frame phrase *647*
ACCUMULATE statement *107*
ActBufferId property *2054*
ActIndexId property *2054*
ActIOFileId property *2054*
ActIOTypeId property *2054*
ACTIVE attribute *1363*
ACTIVE-FORM system reference *109*
ACTIVE-WINDOW system handle
attributes *1258*
ActiveX controls (OCXs)
ADD-EVENTS-PROCEDURE method *1366*
BGCOLOR attribute *1405*
COLUMN attribute *1435*
COM-HANDLE attribute *1439*
COM-SELF system handle *1288*
Control name as property *1459*
CONTROL-FRAME widget *1222*
Controls property *1459*
DYNAMIC attribute *1510*
FRAME attribute *1558*
FRAME-COL attribute *1558*
FRAME-NAME attribute *1558*
FRAME-ROW attribute *1558*
FRAME-X attribute *1559*
FRAME-Y attribute *1559*
Height property *1603*
HEIGHT-CHARS attribute *1604*
HEIGHT-PIXELS attribute *1604*
HELP attribute *1604*
HIDDEN attribute *1605*
HWND attribute *1609*
Left property *1645*
LoadControls method *1651*
MOVE-AFTER-TAB-ITEM method *1690*
MOVE-BEFORE-TAB-ITEM method *1690*
MOVE-TO-BOTTOM method *1692*
MOVE-TO-TOP method *1692*
NAME attribute *1695*
ActiveX controls (OCXs) *(continued)*
Name property *1695*
NEXT-SIBLING attribute *1699*
NEXT-TAB-ITEM attribute *1701*
PARENT attribute *1718*
PREV-SIBLING attribute *1726*
PREV-TAB-ITEM attribute *1727*
PRIVATE-DATA attribute *1730*
REMOVE-EVENTS-PROCEDURE method *1765*
ROW attribute *1784*
SENSITIVE attribute *1806*
TAB-POSITION attribute *1865*
Top property *1875*
TYPE attribute *1880*
VISIBLE attribute *1893*
Widget-Handle property *1896*
Width property *1899*
WIDTH-CHARS attribute *1899*
WIDTH-PIXELS attribute *1900*
X attribute *1929*
Y attribute *1936*
ActiveX controls and Automation objects *1358*
See also COM objects
ActiveX event procedures
DEFINE PARAMETER statement *429*
ACTOR attribute *1363*
ActOtherId property *2055*
ActRecordId property *2055*
ActSpaceId property *2055*
ActSummaryId property *2056*
AdapterType property *2056*
Add additional index entries here (if needed)
Nest index terms for sub entries *835*
ADD-BUFFER method *1364*
ADD-CALC-COLUMN method *1364*
ADD-COLUMNS-FROM method *1366*
ADD-EVENTS-PROCEDURE method *1366*
ADD-FIELDS-FROM method *1367*
ADD-FIRST method *1368*
ADD-HEADER-ENTRY method *1369*
ADD-INDEX-FIELD method *1370*
ADD-INTERVAL function *110*
ADD-LAST method *1370*
ADD-LIKE-COLUMN method *1372*
ADD-LIKE-FIELD method *1373*
ADD-LIKE-INDEX method *1374*
ADD-NEW-FIELD method *1374*
ADD-NEW-INDEX method *1376*
ADD-PARENT-ID-RELATION( ) method *1377*
ADD-RELATION method *1377*
ADD-SCHEMA-LOCATION method *1379*
ADD-SOURCE-BUFFER method *1380*
ADD-SUPER-PROCEDURE method *1380*
ADD-SUPER-PROCEDURE( ) Method *1380*
Add( ) method (JsonArray) *2056*
Add( ) method (JsonObject) *2061*

Add( ) method (Progress.Util.EnumHelper) *2063*
Addition assignment operator (+=) *91*
Addition operator (+) *87*
AddMessage( ) method *2064*
AddNull( ) method (JsonArray) *2064*
AddNull( ) method (JsonObject) *2065*
AddNumber( ) method (JsonArray) *2066*
AddNumber( ) method (JsonObject) *2067*
ADM-DATA attribute *1385*
AFTER option
  INSERT-ROW method *1628*
AFTER-BUFFER attribute *1385*
AFTER-FILL event *1956*
AFTER-ROW-FILL event *1957*
AFTER-ROWID attribute *1385*
AFTER-TABLE attribute *1386*
AfterFetchNextBatch event *2275*
AgentId property *2067*
Aggregate phrase
  ACCUM function *106*
  ACCUMULATE statement *107*
  DISPLAY statement *531*
AGGREGATE statement *113*
ALIAS function *115*
Aliases
  DICTDB *1165*
ALL option
  CLEAR statement *204*
  DISABLE statement *525*
  ENABLE statement *568*
  HIDE statement *710*
  UNSUBSCRIBE statement *1155*
ALLOW-COLUMN-SEARCHING attribute *1386*
ALLOW-PREV-DESERIALIZATION attribute *1386*
ALLOW-REPLICATION option
  DISABLE TRIGGERS statement *528*
AllowEdit property *2068*
AllowNew property *2068*
AllowRemove property *2068*
ALTERNATE-KEY option
  SYSTEM-HELP statement *1115*
ALWAYS option
  SET-REPOSITIONED-ROW method *1836*
ALWAYS-ON-TOP attribute *1387*
AMBIGUOUS attribute *1387*
AMBIGUOUS function *116*
AND operator (bitwise) *117*
AND operator (logical) *118*
AND option
  CAN-FIND function *155*
  Record phrase *985*
And( ) method *2069*
ANSI-ONLY option
  SYSTEM-DIALOG FONT statement *1107*
ANY-KEY event *1943*
ANY-PRINTABLE event *1943*
ANYWHERE
  event procedure names *919*
ANYWHERE option
  ON statement *848*
  SUBSCRIBE statement *1096*
  Trigger phrase *1131*
APPEND fill mode *1540*
APPEND option
  COMPILE statement *222*
  COPY-LOB statement *264*
  DEFINE PARAMETER statement *429*
  OUTPUT TO statement *880*
  Parameter definition syntax *892*
  Parameter passing syntax *898*
Append property *2070*
APPEND-CHILD method *1387*
APPL-ALERT-BOXES attribute *1388*
APPL-CONTEXT-ID attribute *1388*
APPLICATION option
  DDE INITIATE statement *358*
  LOAD statement *785*
APPLY statement *119*
APPLY-CALLBACK method *1389*
APPSERVER-INFO attribute *1389*
APPSERVER-PASSWORD attribute *1390*
APPSERVER-USERID attribute *1390*
ArchiveIndexStatistics property *2070*
ArchiveTableStatistics property *2071*
AreEqual( ) method *2071*
AreNotEqual( ) method *2072*
Argument reference *68*
Array property *2073*
array reference *67*
Array reference ([ ]) *67*
ARRAY-MESSAGE option
  QUERY-TUNING phrase *967*
ArrayIndex property *2073*
Arrays
  ABL and .NET *317*
  assignment *317*
  deep copy *317*
  defined
    ABL *315*
  determinate *317*
  indeterminate *317*
  subscripted references *317*
  unsubscripted references *317*
Arrays, array reference *67*
AS CLASS option
  DEFINE PARAMETER statement *429*
  DEFINE PROPERTY statement *442*
  DEFINE VARIABLE statement *490*
  Parameter definition syntax *892*
AS data types
  defined *317*

AS option
DEFINE PARAMETER statement 429
DEFINE PROPERTY statement 442
DEFINE TEMP-TABLE statement 477
DEFINE VARIABLE statement 490
DEFINE WORK-TABLE statement 504
Format phrase 638
MESSAGE statement 799
Parameter definition syntax 892
Parameter passing syntax (.NET objects) 898
AS PRIMARY option
DEFINE TEMP-TABLE statement 477
ASC function 121
ASCENDING option
DEFINE TEMP-TABLE statement 477
Ascending property 2073
ASK-OVERWRITE option
SYSTEM-DIALOG GET-FILE statement 1110
ASSIGN option
BUFFER-COPY statement 146
CREATE BROWSE statement 278
CREATE SERVER statement 294
CREATE widget statement 299
ASSIGN statement 126
Assign( ) method (Progress.Data.BindingSource) 2074
Assignment (=) statement 122
ASYNC-REQUEST-COUNT attribute 1391
ASYNC-REQUEST-HANDLE attribute 1391
ASYNCHRONOUS attribute 1390
ASYNCHRONOUS option
RUN statement
local procedure call 1023
remote procedure call 1023
Asynchronous request object handle
attributes 1259
events 1259
Asynchronous request object handle events
WAIT-FOR statement 1186
AT option
ENABLE statement 568
PROMPT-FOR statement 929
PUT statement 948
SET statement 1064
UPDATE statement 1157
AT phrase
DEFINE FRAME statement 410
FORM statement 632
Format phrase 638
frame phrase 647
ATTACH-DATA-SOURCE method 1391
ATTACHED-PAIRLIST attribute 1392
ATTR-SPACE attribute 1393
ATTRIBUTE-NAMES attribute 1393
Attributes
handle-based object
accessing 1356
Attributes and methods
handle-based object 1353
Attributes of handles
ACTIVE-WINDOW system handle 1258
Asynchronous request object handle 1259
AUDIT-CONTROL system handle 1261
Buffer object handle 1263
Buffer-field object handle 1265
Call object handle 1266
Client-principal object handle 1273
CLIPBOARD system handle 1279
CODEBASE-LOCATOR system handle 1284
COLOR-TABLE system handle 1285
COMPILER system handle 1288
CURRENT-WINDOW system handle 1290
Data-relation object handle 1291
Data-source object handle 1292
DEBUGGER system handle 1293
DEFAULT-WINDOW system handle 1296
ERROR-STATUS system handle 1299
FILE-INFO system handle 1302
FOCUS system handle 1304
FONT-TABLE system handle 1305
LAST-EVENT system handle 1307
LOG-MANAGER system handle 1297, 1309
procedure object handle 1309
ProDataSet object handle 1311
PROFILER system handle 1313
Query object handle 1315
RCODE-INFO system handle 1316
SAX-attributes object handle 1318
SAX-reader object handle 1319
SAX-writer object handle 1320
SECURITY-POLICY system handle 1321
SELF system handle 1323
Server object handle 1325
Server socket object handle 1326
SESSION system handle 1327
SOAP-fault object handle 1332
SOAP-fault-detail object handle 1333
SOAP-header object handle 1334
Socket object handle 1336
SOURCE-PROCEDURE system handle 1337
TARGET-PROCEDURE system handle 1340
Temp-table object handle 1343
THIS-PROCEDURE system handle 1345
Transaction object handle 1348
WEB-CONTEXT system handle 1349
X-document object handle 1350
X-noderef object handle 1351
Attributes of widgets
BROWSE widget 1208
BUTTON widget 1218
COMBO-BOX widget 1220
CONTROL-FRAME widget 1222
DIALOG-BOX widget 1225

Attributes of widgets *(continued)*
EDITOR widget *1227*
FIELD-GROUP widget *1230*
FILL-IN widget *1231*
FRAME widget *1233*
IMAGE widget *1237*
LITERAL widget *1238*
MENU widget *1239*
MENU-ITEM widget *1240*
RADIO-SET widget *1241*
RECTANGLE widget *1243*
SELECTION-LIST widget *1244*
SHADOW-WINDOW widget *1246*
SLIDER widget *1247*
SUB-MENU widget *1249*
TEXT widget *1250*
TOGGLE-BOX widget *1251*
WINDOW widget *1253*
AUDIT-CONTROL system handle
attributes *1261*
methods *1261*
AUDIT-ENABLED function *134*
AUDIT-EVENT-CONTEXT attribute *1394*
AUDIT-POLICY system handle
methods *1262*
AUTHENTICATION-FAILED method *1394*
AuthenticationAccess property *2074*
AuthenticationFailed property *2074*
AUTO-COMPLETION attribute *1395*
AUTO-COMPLETION option
COMBO-BOX phrase *215*
AUTO-DELETE attribute *1395*
AUTO-DELETE-XML attribute *1396*
AUTO-END-KEY attribute *1396*
AUTO-END-KEY option
DEFINE BUTTON statement *386*
AUTO-GO attribute *1396*
AUTO-GO option
DEFINE BUTTON statement *386*
AUTO-INDENT attribute *1396*
AUTO-RESIZE attribute *1397*
AUTO-RETURN attribute *1398*
AUTO-RETURN option
CHOOSE statement *168*
DEFINE BROWSE statement *366*
Format phrase *638*
MESSAGE statement *799*
AUTO-SYNCHRONIZE attribute *1398*
AUTO-VALIDATE attribute *1398*
AUTO-ZAP attribute *1399*
Automation object access *274*
Automation Servers launched
CREATE automation object statement *274*
AutoSort property *2074*
AutoSync property *2075*
AutoUpdate property *2076*
AVAILABLE attribute *1399*
AVAILABLE function *135*
AVAILABLE-FORMATS attribute *1400*
AvailableBytes( ) method *2076*
AVERAGE option
aggregate phrase *111*

B

BACK-TAB event *1941*
BACKGROUND attribute *1400*
BACKGROUND option
DEFINE FRAME statement *410*
FORM statement *632*
Backslash (\)
special character *64*
BACKSPACE event *1943*
BACKWARDS option
REPOSITION statement *1008*
BandIndex property *2077*
BASE-ADE attribute *1400*
BASE-KEY option
LOAD statement *785*
BASE64-DECODE function *136*
BASE64-ENCODE function *137*
BASIC-LOGGING attribute *1401*
Batch mode
events *1186*
BATCH-MODE attribute *1402*
BATCH-MODE preprocessor name *74*
BATCH-SIZE attribute *1402*
Batching property *2078*
BEFORE option
INSERT-ROW method *1628*
BEFORE-BUFFER attribute *1402*
BEFORE-FILL event *1956*
BEFORE-HIDE option
PAUSE statement *912*
BEFORE-ROW-FILL event *1957*
BEFORE-ROWID attribute *1403*
BEFORE-TABLE attribute *1403*
BEFORE-TABLE option
DEFINE TEMP-TABLE statement *477*
BeforeFetchNextBatch event *2276*
BEGIN-EVENT-GROUP method *1403*
BEGINS operator *138*
BELL event *1941*
BELL statement *139*
BGCOLOR attribute *1405*
BGCOLOR option
DEFINE BROWSE statement *366*
DEFINE BUTTON statement *386*
DEFINE FRAME statement *410*
DEFINE IMAGE statement *419*
DEFINE MENU statement *423*
DEFINE RECTANGLE statement *465*

BGCOLOR option *(continued)*
DEFINE SUB-MENU statement *471*
DEFINE VARIABLE statement *490*
ENABLE statement *568*
FORM statement *632*
Format phrase *638*
frame phrase *647*
Frame phrase *647*
PROMPT-FOR statement *929*
BINARY option
BUFFER-COMPARE statement *144*
INPUT FROM statement *727*
OUTPUT TO statement *880*
BIND option
DEFINE PARAMETER statement *429*
Parameter definition syntax *892*
Parameter passing syntax *898*
BIND-WHERE option
QUERY-TUNING phrase *967*
BLANK attribute *1405*
BLANK option
Format phrase *638*
blb files
EXPORT statement *594*
BLINK option
COLOR phrase *210*
BLOB data type *318*
BLOCK-ITERATION-DISPLAY attribute *1405*
Block-level
error redirection *140*
BLOCK-LEVEL ON ERROR UNDO, THROW statement *140*
BlockId property *2079*
Boolean property *2079*
BORDER-BOTTOM-CHARS attribute *1406*
BORDER-BOTTOM-PIXELS attribute *1406*
BORDER-LEFT-CHARS attribute *1406*
BORDER-LEFT-PIXELS attribute *1406*
BORDER-RIGHT-CHARS attribute *1407*
BORDER-RIGHT-PIXELS attribute *1407*
BORDER-TOP-CHARS attribute *1407*
BORDER-TOP-PIXELS attribute *1407*
BOX attribute *1407*
BOX function *142*
BOX-SELECTABLE attribute *1408*
Boxing
defined *325*
BREAK option
FOR statement *620*
OPEN QUERY statement *857*
PRESELECT phrase *914*
BRIGHT option
COLOR phrase *210*
Browse cell
applying events *1938*
referencing *1208*
Browse column
referencing *1208*
Browse enable phrase
DEFINE BROWSE statement *366*
BROWSE option
ASSIGN statement *126*
DEFINE BROWSE statement *366*
Browse options phrase
DEFINE BROWSE statement *366*
BROWSE widget
applying events *1938*
attributes *1208*
events *1208*
methods *1208*
referencing *1208*
Buffer object
CREATE BUFFER statement *281*
Buffer object handle
attributes *1263*
events *1263*
methods *1263*
BUFFER option
DEFINE BUFFER statement *380*
LDBNAME function *770*
Parameter definition syntax *892*
Parameter passing syntax *898*
RAW-TRANSFER statement *980*
BUFFER-CHARS attribute *1408*
BUFFER-CHARS option
EDITOR phrase *564*
BUFFER-COMPARE method *1408*
BUFFER-COMPARE statement *144*
BUFFER-COPY method *1410*
BUFFER-COPY statement *146*
BUFFER-CREATE method *1412*
BUFFER-DELETE method *1412*
BUFFER-FIELD attribute *1413*
BUFFER-FIELD method *1413*
Buffer-field object handle
attributes *1265*
BUFFER-GROUP-ID attribute *1413*
BUFFER-GROUP-ID function *148*
BUFFER-GROUP-NAME attribute *1414*
BUFFER-GROUP-NAME function *149*
BUFFER-HANDLE attribute *1414*
BUFFER-LINES attribute *1414*
BUFFER-LINES option
EDITOR phrase *564*
BUFFER-NAME attribute *1415*
BUFFER-NAME option
CREATE-BUFFER statement *281*
BUFFER-PARTITION-ID attribute *1415*
BUFFER-PARTITION-ID function *149*
BUFFER-RELEASE method *1415*
BUFFER-TENANT-ID attribute *1416*
BUFFER-TENANT-ID function *150*

BUFFER-TENANT-NAME attribute *1416*
BUFFER-TENANT-NAME function *151*
BUFFER-VALIDATE method *1417*
BUFFER-VALUE method *1417*
BufferHdl property *2079*
BufferName property *2079*
BuffStatusId property *2080*
Building menus
    CHOOSE statement *168*
Built-in class events *2275*
BUTTON option
    DEFINE BUTTON statement *386*
    MESSAGE statement *799*
BUTTON widget
    attributes *1218*
    dynamic *299*
    events *1218*
    methods *1218*
BY option
    aggregate phrase *111*
    FOR statement *620*
    OPEN QUERY statement *857*
    PRESELECT phrase *914*
    REPEAT statement *1001*
BY-REFERENCE option
    Parameter passing syntax *898*
BY-VALUE option
    DEFINE PARAMETER statement *429*
    Parameter definition syntax *892*
    Parameter passing syntax *898*
BYTE parameter data type *429*
Byte swapping
    GET-DOUBLE function *690*, *692*
    GET-FLOAT function *694*
    GET-LONG function *699*
    GET-SHORT function *701*
    GET-UNSIGNED-LONG function *703*
    GET-UNSIGNED-SHORT function *704*
    PUT-DOUBLE statement *955*
    PUT-FLOAT statement *956*
    PUT-LONG statement *962*
    PUT-SHORT statement *963*
    PUT-UNSIGNED-LONG statement *965*
    PUT-UNSIGNED-SHORT statement *966*
BYTES-READ attribute *1418*
BYTES-WRITTEN attribute *1418*
BytesWritten property *2080*

C

C option
    Character-string literal (" ") *67*
Cache
    schema *1047*
CACHE attribute *1419*
CACHE option
    DEFINE QUERY statement *460*
CACHE-SIZE option
    QUERY-TUNING phrase *967*
Call object
    creating *283*
Call object handle
    attributes *1266*
    methods *1266*
CALL statement *151*
CALL-NAME attribute *1419*
CALL-TYPE attribute *1420*
CallStack property *2080*
CAN-CREATE attribute *1424*
CAN-DELETE attribute *1425*
CAN-DO function *152*
CAN-DO-DOMAIN-SUPPORT attribute *1425*
CAN-FIND function *155*
CAN-QUERY function *158*
CAN-READ attribute *1425*
CAN-SET function *159*
CAN-WRITE attribute *1426*
CANCEL-BREAK method *1421*
CANCEL-BUTTON attribute *1422*
CANCEL-BUTTON option
    frame phrase *647*
CANCEL-REQUESTS method *1422*
CANCEL-REQUESTS-AFTER method *1423*
CancelCreateRow event *2277*
CANCELLED attribute *1424*
CanRead property *2082*
CanWrite property *2082*
CAPS function *160*
CAREFUL-PAINT attribute *1426*
Case sensitivity
    WHERE option *985*
CASE statement *161*
CASE-SENSITIVE attribute *1426*
CASE-SENSITIVE option
    BUFFER-COMPARE statement *144*
    DEFINE PARAMETER statement *429*
    DEFINE VARIABLE statement *490*
CAST function *163*
Catch block
    FOR statement *620*
CATCH statement *164*
CDECL option
    PROCEDURE statement *919*
CENTERED attribute *1426*
CENTERED option
    frame phrase *647*
Chain
    Super procedure *1380*
Chained handle references *1357*

CHAINED option
TRANSACTION-MODE AUTOMATIC statement *1130*
CHAR data type *318*
Character conversions
assignment (=) statement *122*
CHARACTER data type *318*
CHARACTER parameter data type *429*
Character-string literal (" ") *67*
CHARSET attribute *1427*
CHECKED attribute *1427*
Child frames
example *410*
specifying *632*
CHILD-BUFFER attribute *1427*
CHILD-NUM attribute *1427*
ChildAllowEdit property *2083*
ChildAllowNew property *2083*
ChildAllowRemove property *2084*
ChildInputValue property *2084*
ChildrenAllowEdit class *1970*
ChildrenAllowNew class *1970*
ChildrenAllowRemove class *1971*
ChildrenInputValue class *1971*
CHOOSE event *1947*
CHOOSE statement *168*
CHR function *173*
Class body
CLASS statement *185*
Class events
built-in *2275*
defining *2272*
defining handlers *2273*
using *2274*
Class members *315*
Class names *1140*
CLASS option
DEFINE EVENT statement *404*
CLASS statement *185*
Class-based data member access *174*
Class-based method call *177*
Class-based object reference *180*
Class-based objects *315*
Class-based property access *181*
CLASS-TYPE attribute *1428*
Classes
built-in *1961*
ChildrenAllowEdit *1970*
ChildrenAllowNew *1970*
ChildrenAllowRemove *1971*
ChildrenInputValue *1971*
Progress.Data.BindingSource *1963*
Progress.Data.CancelCreateRowEventArgs *1969*
Progress.Data.ColumnPropDesc *1971*
Progress.Data.CreateRowEventArgs *1972*
Progress.Data.DataType enumeration *1973*
Classes *(continued)*
Progress.Data.DummyRow *1974*
Progress.Data.InputValue *1975*
Progress.Data.OffEndEventArgs *1976*
Progress.Data.SortRequestEventArgs *1977*
Progress.Data.TableDesc *1978*
Progress.Database.TempTableInfo *1979*
Progress.Database.VSTTableId *1980*
Progress.IO.BinarySerializer *1981*
Progress.IO.ByteOrder enumeration *1982*
Progress.IO.FileInputStream *1982*, *1986*
Progress.IO.FileOutputStream *1983*
Progress.IO.InputStream *1984*
Progress.IO.JsonSerializer *1985*
Progress.IO.MemoryOutputStream *1989*
Progress.IO.OutputStream *1992*
Progress.Json.JsonError *1992*
Progress.Json.JsonParser *1993*
Progress.Json.JsonParserError *1994*
Progress.Json.ObjectModel.JsonArray *1994*
Progress.Json.ObjectModel.JsonConstruct *1996*
Progress.Json.ObjectModel.JsonDataType *1997*
Progress.Json.ObjectModel.JsonObject *1998*
Progress.Json.ObjectModel.ObjectModelParser *1999*
Progress.Lang.AppError *2000*
Progress.Lang.Class *2002*
Progress.Lang.Enum *2005*
Progress.Lang.FlagsEnum *2006*
Progress.Lang.LockConflict *2007*
Progress.Lang.Object *2008*
Progress.Lang.ParameterList *2011*
Progress.Lang.ProError *2012*
Progress.Lang.SoapFaultError *2013*
Progress.Lang.Stop *2014*
Progress.Lang.StopAfter *2015*
Progress.Lang.SysError *2016–2017*
Progress.Lang.UserInterrupt *2018*
Progress.Reflect.Constructor *2020*
Progress.Reflect.Event *2022*
Progress.Reflect.Method *2024*
Progress.Reflect.Parameter *2025*
Progress.Reflect.Property *2026*
Progress.Reflect.Variable *2026*
Progress.Security.PAMStatus *2027*
Progress.Util.DateTimeHelper *2028*
Progress.Util.DecimalHelper *2029*
Progress.Util.EnumHelper *2030*
Progress.Util.ResourceHelper *2031*
Progress.Util.StringHelper *2032*
Progress.Util.TypeHelper *2033*
Progress.Windows.Form *2034*
Progress.Windows.FormProxy *2038*
Progress.Windows.MDIChildForm *2040*
Progress.Windows.MDIChildRadForm *2041*
Progress.Windows.MDIChildRadRibbonForm *2042*
Progress.Windows.MDIChildShapedForm *2043*

Classes *(continued)*
Progress.Windows.OERadForm *2043*
Progress.Windows.OERadRibbonForm *2044*
Progress.Windows.OEShapedForm *2044*
Progress.Windows.UserControl *2045*
Progress.Windows.WindowContainer *2048*
System.Exception *2049*
user-defined *185*
CLEAR event *1943*
CLEAR method (Call Object Handle) *1428*
CLEAR method (DEBUGGER Handle) *1428*
CLEAR method (FILL-IN Widget) *1429*
CLEAR method (ProDataSet Object Handle) *1429*
CLEAR method (Temp-Table Object Handle) *1430*
CLEAR option
SHOW-STATS statement *1083*
CLEAR statement *204*
CLEAR-APPL-CONTEXT method *1430*
CLEAR-LOG method *1431*
CLEAR-SELECTION method *1431*
CLEAR-SORT-ARROWS method *1432*
Clear( ) method (Class) *2087*
Clear( ) method (MemoryOutputStream) *2087*
CLIENT-CONNECTION-ID attribute *1432*
Client-principal object handle
attributes *1273*
methods *1273*
CLIENT-TTY attribute *1433*
CLIENT-TYPE attribute *1433*
CLIENT-WORKSTATION attribute *1433*
ClientContextId property *2085*
CLIPBOARD option
OUTPUT TO statement *880*
CLIPBOARD system handle
attributes *1279*
CLOB data type *318*
CLONE-NODE method *1434*
Clone( ) method (JsonArray) *2088*
Clone( ) method (JsonObject) *2088*
Clone( ) method (Progress.Lang.Object) *2089*
CLOSE event *1954*
CLOSE QUERY statement *205*
CLOSE STORED-PROCEDURE statement *207*
CLOSE-LOG method *1434*
Close( ) method (IO) *2089*
Closed property *2090*
CODE attribute *1435*
CODEBASE-LOCATOR system handle
attributes *1284*
CODEPAGE attribute *1435*
CODEPAGE-CONVERT function *208*
COLLATE option
FOR statement *620*
OPEN QUERY statement *857*
OUTPUT TO statement *880*
PRESELECT phrase *914*
Colon (:)
punctuation *61*
COLON option
format phrase *638*
COLON-ALIGNED option
AT phrase *131*
COLOR option
CHOOSE statement *168*
frame phrase *647*
PUT SCREEN statement *946*
PUT-KEY-VALUE statement *958*
COLOR phrase
COLOR statement *213*
DOS colors *240*
MESSAGE statement *799*
COLOR statement *213*
COLOR-TABLE system handle
attributes *1285*
methods *1285*
COLUMN attribute *1435*
COLUMN option
AT phrase *131*
frame phrase *647*
PUT CURSOR statement *944*
PUT SCREEN statement *946*
COLUMN-BGCOLOR attribute *1436*
COLUMN-CODEPAGE option
DEFINE TEMP-TABLE statement *477*
COLUMN-DCOLOR attribute *1436*
COLUMN-FGCOLOR attribute *1436*
COLUMN-FONT attribute *1437*
COLUMN-LABEL attribute *1437*
COLUMN-LABEL option
DEFINE PARAMETER statement *429*
DEFINE VARIABLE statement *490*
format phrase *638*
COLUMN-MOVABLE attribute *1437*
COLUMN-OF option
AT phrase *131*
COLUMN-PFCOLOR attribute *1438*
COLUMN-READ-ONLY attribute *1438*
COLUMN-RESIZABLE attribute *1438*
COLUMN-SCROLLING attribute *1439*
COLUMNS option
frame phrase *647*
COM object properties
CONTROL-FRAME widget *1222*
COM objects
referencing *1358*
releasing *999*
COM-HANDLE attribute *1439*
COM-HANDLE data type *318*
COM-SELF system handle *1288*
COMBO-BOX phrase
VIEW-AS phrase *1180*

COMBO-BOX widget
attributes 1220
dynamic 299
events 1220
methods 1220
Comma (,)
punctuation 63
COMMAND option
DDE EXECUTE statement 354
SYSTEM-HELP statement 1115
Comments
multi-line 84
single-line 85
COMPARE function 220
CompareTo( ) method 2090
COMPILE statement 222
Compile-time arguments
RUN statement 1023
COMPILER system handle
attributes 1288
methods 1288
Compiling with NO-LOCK 222
Complement( ) method 2091
COMPLETE attribute 1439
COMPLETE option
SAVE CACHE statement 1047
Complex types 319
Concatenation operator (+) 88
CONDITIONAL option
SET-REPOSITIONED-ROW method 1836
Conditions
asynchronous event procedures 1023, 1186
RUN statement 1023
CONFIG-NAME attribute 1440
CONNECT event 1955
CONNECT method
AppServer 1440
Socket object 1449
Web service 1452
CONNECT option
CREATE automation object statement 274–275
CONNECT statement 253
CONNECT TO option by file extension
CREATE automation object statement 276
CONNECT TO option per file
CREATE automation object statement 275
CONNECTED function 259
CONNECTED method 1457
Constant value
Record phrase 985
Constructed generic types 317
Constructor body
CONSTRUCTOR statement 260
CONSTRUCTOR statement 260
CONTAINS operator 985
CONTENTS option
SYSTEM-HELP statement 1115
CONTEXT option
SYSTEM-HELP statement 1115
CONTEXT-HELP attribute 1457
CONTEXT-HELP option
frame phrase 647
CONTEXT-HELP-FILE attribute 1458
CONTEXT-HELP-FILE option
frame phrase 647
CONTEXT-HELP-ID attribute 1458
CONTEXT-HELP-ID option
DEFINE BROWSE statement 366
DEFINE BUTTON statement 386
DEFINE VARIABLE statement 490
CONTEXT-POPUP option
SYSTEM-HELP statement 1115
Control name as property 1459
CONTROL option
PUT statement 948
CONTROL-BOX attribute 1458
CONTROL-FRAME widget
attributes 1222
COM object properties 1222
dynamic 299
events 1222
methods 1222
Controls property 1459
CONVERT option
COPY-LOB statement 264
INPUT FROM statement 727
INPUT THROUGH statement 733
INPUT-OUTPUT THROUGH statement 738
OUTPUT THROUGH statement 876
OUTPUT TO statement 880
CONVERT-3D-COLORS attribute 1460
CONVERT-3D-COLORS option
DEFINE IMAGE statement 419
CONVERT-TO-OFFSET method 1460
COPY-DATASET method 1461
COPY-LOB statement 264
COPY-SAX-ATTRIBUTES method 1463
COPY-TEMP-TABLE method 1464
COUNT option
aggregate phrase 111
Count property 2092
COUNT-OF function 268
COUNT-OF keyword value
DBRESTRICTIONS function 347
COVERAGE attribute 1466
CPCASE attribute 1467
CPCOLL attribute 1467
CPINTERNAL attribute 1467
CPLOG attribute 1468
CPPRINT attribute 1468
CPRCODEIN attribute 1468

CPRCODEOUT attribute *1468*
CPSTREAM attribute *1468*
CPTERM attribute *1469*
CRC-VALUE attribute *1469*
CREATE ALIAS statement *271*
CREATE automation object statement *274*
CREATE BROWSE statement *278*
CREATE BUFFER statement *281*
CREATE CALL statement *283*
CREATE CLIENT-PRINCIPAL statement *284*
CREATE DATA-SOURCE statement *288*
CREATE DATABASE statement *285*
CREATE DATASET statement *287*
CREATE QUERY statement *289*
CREATE SAX-ATTRIBUTES statement *290*
CREATE SAX-READER statement *293*
CREATE SAX-WRITER statement *294*
CREATE SERVER statement *294*
CREATE SERVER-SOCKET statement *295*
CREATE SOAP-HEADER statement *295*
CREATE SOAP-HEADER-ENTRYREF statement *296*
CREATE SOCKET statement *297*
CREATE statement *268*
CREATE TEMP-TABLE statement *297*
CREATE widget statement *299*
CREATE WIDGET-POOL statement *302*
CREATE X-DOCUMENT statement *304*
CREATE X-NODEREF statement *305*
CREATE-LIKE method *1470*
CREATE-LIKE-SEQUENTIAL method *1471*
CREATE-NODE method *1472*
CREATE-NODE-NAMESPACE method *1473*
CREATE-RESULT-LIST-ENTRY method *1475*
CREATE-TEST-FILE option
  SYSTEM-DIALOG GET-FILE statement *1110*
Create( ) method (CHARACTER) *2093*
Create( ) method (DATETIME) *2094*
Create( ) method (DECIMAL) *2097*
Created property *2099*
CreateRow event *2277*
CURRENT option
  FIND statement *609*
  GET statement *682*
  SAVE CACHE statement *1047*
Current working directory *1203*
CURRENT-CHANGED attribute *1475*
CURRENT-CHANGED function *306*
CURRENT-COLUMN attribute *1475*
CURRENT-ENVIRONMENT attribute *1476*
CURRENT-ITERATION attribute (Data Objects) *1476*
CURRENT-ITERATION attribute (Widget Objects) *1476*
CURRENT-LANGUAGE function *308*
CURRENT-LANGUAGE statement *309*
CURRENT-QUERY method *1477*
CURRENT-REQUEST-INFO attribute *1479*
CURRENT-RESPONSE-INFO attribute *1480*
CURRENT-RESULT-ROW attribute *1481*
CURRENT-RESULT-ROW function *310*
CURRENT-ROW-MODIFIED attribute *1481*
CURRENT-VALUE function *311*
CURRENT-VALUE statement *313*
CURRENT-WINDOW attribute *1481*
CURRENT-WINDOW system handle
  attributes *1290*
Cursor indicator
  described *609*
CURSOR-CHAR attribute *1482*
CURSOR-DOWN event *1941*
CURSOR-LEFT event *1942*
CURSOR-LINE attribute *1482*
CURSOR-OFFSET attribute *1482*
CURSOR-RIGHT event *1942*
CURSOR-UP event *1942*
Custom property *2099*

D

Data member
  accessing *174*
Data property *2099*
Data types
  AS data types *317*
  BLOB *318*
  CHAR *318*
  CHARACTER *318*
  CLOB *318*
  COM-HANDLE *318*
  complex *319*
  .NET generic *317*
  .NET *315*
  DATE *318*
  DATETIME *318*
  DATETIME-TZ *318*
  DECIMAL *318*
  Default display formats *799*
  HANDLE *318*
  INT *318*
  INT64 *318*
  INTEGER *318*
  LOGICAL *318*
  LONGCHAR *318*
  MEMPTR *319*
  RAW *319*
  RECID *319*
  ROWID *319*
  scalar *320*
  user-defined
    class *185*
    interface *747*
    naming *1140*
  WIDGET-HANDLE *319*
DATA-ENTRY-RETURN attribute *1483*

Data-relation object handle
attributes *1291*
DATA-RELATION option
DEFINE DATASET statement *393*
DATA-SOURCE attribute *1483*
Data-source object handle
attributes *1292*
methods *1292*
DATA-SOURCE option
DEFINE DATA-SOURCE statement *401*
DATA-SOURCE-COMPLETE-MAP attribute *1483*
DATA-SOURCE-MODIFIED attribute *1484*
DATA-SOURCE-MODIFIED function *335*
DATA-SOURCE-ROWID attribute *1484*
DATA-TYPE attribute *1485*
DataServers
DATASERVERS function *334*
DBRESTRICTIONS function *347*
GATEWAYS function *678*
RUN STORED-PROCEDURE statement *1023*
stored procedures *1023*
DATASERVERS function *334*
DATASET attribute *1485*
DATASET option
DEFINE PARAMETER statement *429*
Parameter definition syntax *892*
Parameter passing syntax *898*
DATASET-HANDLE option
DEFINE PARAMETER statement *429*
Parameter definition syntax *892*
Parameter passing syntax *898*
DataType property *2100*
DataTypeName property (Reflect) *2100*
Date addition operator (+) *89*
DATE data type *318*
DATE function *335*
Date subtraction operator (-) *95*
DATE-FORMAT attribute *1485*
Datetime addition operator (+) *90*
DATETIME data type *318*
DATETIME function *338*
Datetime subtraction operator (-) *96*
DATETIME-TZ data type *318*
DATETIME-TZ function *340*
DAY function *342*
DB-CONTEXT attribute *1486*
DB-LIST attribute *1486*
DB-REFERENCES attribute *1487*
DB-REMOTE-HOST function *347*
DBCODEPAGE function *343*
DBCOLLATION function *344*
DBNAME attribute *1487*
DBNAME function *345*
DBPARAM function *346*
DBRESTRICTIONS function *347*
DbStatusId property *2100*
DBTASKID function *349*
DBTYPE function *350*
DBVERSION function *351*
DCOLOR attribute *1487*
DCOLOR option
DEFINE BROWSE statement *366*
DEFINE BUTTON statement *386*
DEFINE FRAME statement *410*
DEFINE MENU statement *423*
DEFINE RECTANGLE statement *465*
DEFINE SUB-MENU statement *471*
DEFINE VARIABLE statement *490*
ENABLE statement *568*
FORM statement *632*
format phrase *638*
frame phrase *647*
DDE ADVISE statement *352*
DDE EXECUTE statement *354*
DDE GET statement *356*
DDE INITIATE statement *358*
DDE REQUEST statement *359*
DDE SEND statement *361*
DDE TERMINATE statement *363*
DDE-ERROR attribute *1488*
DDE-ID attribute *1489*
DDE-ITEM attribute *1489*
DDE-NAME attribute *1489*
DDE-NOTIFY event *1937*
DDE-TOPIC attribute *1490*
DEBLANK attribute *1490*
DEBLANK option
format phrase *638*
DEBUG method *1490*
DEBUG option
QUERY-TUNING phrase *967*
DEBUG-ALERT attribute *1491*
DEBUG-LIST option
COMPILE statement *222*
DEBUGGER system handle
attributes *1293*
methods *1293*
DECIMAL data type *318*
DECIMAL function *364*
DECIMALS attribute *1491*
DECIMALS option
DEFINE PARAMETER statement *429*
DEFINE VARIABLE statement *490*
DECLARE-NAMESPACE method *1492*
DeclaringClass property *2101*
DECRYPT function *365*
DEFAULT attribute *1492*
Default display formats
format phrase *638*
MESSAGE statement *799*
PUT statement *948*
Default keyboard events *1943*

Default matching types *317*
DEFAULT option
  DEFINE BUTTON statement *386*
  GET-KEY-VALUE statement *696*
  PUT-KEY-VALUE statement *958*
  STATUS statement *1089*
DEFAULT-ACTION event *1947*
DEFAULT-BUFFER-HANDLE attribute *1493*
DEFAULT-BUTTON attribute *1493*
DEFAULT-BUTTON option
  frame phrase *647*
DEFAULT-COMMIT attribute *1494*
DEFAULT-EXTENSION option
  SYSTEM-DIALOG GET-FILE statement *1110*
DEFAULT-STRING attribute *1494*
DEFAULT-VALUE attribute *1494*
DEFAULT-WINDOW system handle
  attributes *1296*
DEFINE BROWSE statement *366*
DEFINE BUFFER statement *380*
DEFINE BUTTON statement *386*
DEFINE DATA-SOURCE statement *401*
DEFINE DATASET statement *393*
DEFINE ENUM statement *404*
DEFINE EVENT statement *404*
DEFINE FRAME statement *410*
DEFINE IMAGE statement *419*
DEFINE MENU statement *423*
DEFINE PARAMETER statement *429*
DEFINE PROPERTY statement *442*
DEFINE QUERY statement *460*
DEFINE RECTANGLE statement *465*
DEFINE STREAM statement *469*
DEFINE SUB-MENU statement *471*
DEFINE TEMP-TABLE statement *477*
DEFINE VARIABLE statement *490*
DEFINE WORK-TABLE statement *504*
DEFINE WORKFILE statement *509*
DEFINED preprocessor function *509*
Defining parameters *892*
DELEGATE option
  DEFINE EVENT statement *404*
DELETE ALIAS statement *513*
DELETE method *1495*
DELETE OBJECT statement *513*
DELETE PROCEDURE statement *517*
DELETE statement *510*
DELETE WIDGET statement *519*
DELETE WIDGET-POOL statement *521*
DELETE-CHAR method *1496*
DELETE-CHARACTER event *1943*
DELETE-CURRENT-ROW method *1496*
DELETE-HEADER-ENTRY method *1497*
DELETE-LINE method *1497*
DELETE-NODE method *1497*
DELETE-RESULT-LIST-ENTRY method *1498*
DELETE-SELECTED-ROW method *1499*
DELETE-SELECTED-ROWS method *1499*
DELIMITER attribute *1500*
DELIMITER option
  EXPORT statement *594*
  IMPORT statement *717*
DESCENDING option
  DEFINE TEMP-TABLE statement *477*
  FOR statement *620*
  OPEN QUERY statement *857*
  PRESELECT phrase *914*
DESCRIPTION attribute *1500*
DESELECT-FOCUSED-ROW method *1500*
DESELECT-ROWS method *1500*
DESELECT-SELECTED-ROW method *1501*
DESELECTION event *1952*
Deserialize( ) method (BinarySerializer) *2101*
Deserialize( ) method (JsonSerializer) *2102*
Destructor body
  DESTRUCTOR statement *523*
DESTRUCTOR statement *523*
DETACH-DATA-SOURCE method *1501*
Determinate arrays *317*
Developer events *1954*
Device property *2103*
DIALOG-BOX widget
  attributes *1225*
  dynamic *299*
  events *1225*
  methods *1225*
DICTDB alias *1165*
DICTIONARY statement *524*
DIR option
  LOAD statement *785*
Direct manipulation events
  frame-only *1953*
  general *1952*
DIRECTORY attribute *1501*
DISABLE method *1502*
DISABLE statement *525*
DISABLE TRIGGERS statement *528*
DISABLE-AUTO-ZAP attribute *1502*
DISABLE-AUTO-ZAP option
  DEFINE BROWSE statement *366*
  format phrase *638*
DISABLE-CONNECTIONS method *1502*
DISABLE-DUMP-TRIGGERS method *1503*
DISABLE-LOAD-TRIGGERS method *1503*
DISABLED option
  DEFINE MENU statement *423*
  DEFINE SUB-MENU statement *471*
DISCONNECT method (Handle) *1504*
DISCONNECT statement *530*
Display formats
  data type defaults *799*

DISPLAY option
COLOR statement 213
DEFINE BROWSE statement 366
frame phrase 647
DISPLAY preprocessor name 74
DISPLAY statement 531
DISPLAY-MESSAGE method 1504
DISPLAY-TIMEZONE attribute 1505
DISPLAY-TYPE attribute 1505
Dispose( ) method 2104
DisposeDialogOnClose property 2104
Division assignment operator (/=) 103
Division operator (/) 102
DO statement 537
DOMAIN-DESCRIPTION attribute 1506
DOMAIN-NAME attribute 1506
DOMAIN-TYPE attribute 1507
DOS statement 543
dos-hex attributes 210
Double colon (::)
punctuation 62
DOUBLE parameter data type 429
Double quote (")
special character 66
Double-byte enabled
APPLY statement 119
ASC function 121
BEGINS operator 138
CAPS function 160
CHR function 173
ENCODE function 573
ENTRY function 577
ENTRY statement 579
FILL function 605
INDEX function 722
IS-LEAD-BYTE function 759
LASTKEY function 767
LC function 769
LEFT-TRIM function 774
LENGTH function 775
LOOKUP function 793
MATCHES operator 795
NUM-ENTRIES function 837
OVERLAY statement 888
R-INDEX function 973
SEARCH function 1056
STRING function 1094
SUBSTITUTE function 1098
SUBSTRING function 1099
SUBSTRING statement 1100
TRIM function 1137
DOWN attribute 1508
DOWN event 1941
DOWN option
DEFINE BROWSE statement 366
frame phrase 647
DOWN option *(continued)*
SCROLL statement 1049
DOWN statement 544
DRAG-ENABLED attribute 1508
DROP-DOWN option
COMBO-BOX phrase 215
DROP-DOWN-LIST option
COMBO-BOX phrase 215
DROP-FILE-NOTIFY event 1947
DROP-TARGET attribute 1509
DROP-TARGET option
DEFINE BROWSE statement 366
DEFINE BUTTON statement 386
DEFINE VARIABLE statement 490
frame phrase 647
DSLOG-MANAGER system handle
attributes 1297
methods 1297
DUMP option
DISABLE TRIGGERS statement 528
DUMP-LOGGING-NOW method 1509
DYNAMIC attribute 1510
Dynamic browse
ADD-CALC-COLUMN method 1364
ADD-COLUMNS-FROM method 1366
ADD-LIKE-COLUMN method 1372
Dynamic browser
CREATE BROWSE statement 278
Dynamic link library routines
See also shared library routines
ActiveX event procedures
DEFINE PARAMETER statement 429
APPEND option
DEFINE PARAMETER statement 429
AS CLASS option
DEFINE PARAMETER statement 429
AS option
DEFINE PARAMETER statement 429
BIND option
DEFINE PARAMETER statement 429
BY-VALUE option
DEFINE PARAMETER statement 429
CASE-SENSITIVE option
DEFINE PARAMETER statement 429
COLUMN-LABEL option
DEFINE PARAMETER statement 429
DATASET option
DEFINE PARAMETER statement 429
DATASET-HANDLE option
DEFINE PARAMETER statement 429
DECIMALS option
DEFINE PARAMETER statement 429
DEFINE PARAMETER statement 429
EXTENT option
DEFINE VARIABLE statement 429

Dynamic link library routines *(continued)*
FORMAT option
DEFINE PARAMETER statement *429*
GET-BYTE function *685*
GET-DOUBLE function *690*, *692*
GET-FLOAT function *694*
GET-LONG function *699*
GET-POINTER-VALUE function *699*
GET-SHORT function *701*
GET-SIZE function *701*
GET-STRING function *703*
GET-UNSIGNED-LONG function *703*
GET-UNSIGNED-SHORT function *704*
INITIAL option
DEFINE PARAMETER statement *429*
INPUT PARAMETER option
DEFINE PARAMETER statement *429*
INPUT-OUTPUT PARAMETER option
DEFINE PARAMETER statement *429*
LABEL option
DEFINE PARAMETER statement *429*
LIKE option
DEFINE PARAMETER statement *429*
NO-UNDO option
DEFINE PARAMETER statement *429*
OUTPUT PARAMETER option
DEFINE PARAMETER statement *429*
PARAMETER BUFFER option
DEFINE PARAMETER statement *429*
PRESELECT option
DEFINE PARAMETER statement *429*
PROCEDURE statement *919*
PUT-BYTE statement *953*
PUT-BYTES statement *955*
PUT-DOUBLE statement *955*
PUT-FLOAT statement *956*
PUT-LONG statement *962*
PUT-SHORT statement *963*
PUT-STRING statement *964*
PUT-UNSIGNED-LONG statement *965*
PUT-UNSIGNED-SHORT statement *966*
RETURN PARAMETER option
DEFINE PARAMETER statement *429*
RUN statement *1023*
SET-POINTER-VALUE statement *1077*
SET-SIZE statement *1078*
TABLE option
DEFINE PARAMETER statement *429*
TABLE-HANDLE option
DEFINE PARAMETER statement *429*
TEMP-TABLE option
DEFINE PARAMETER statement *429*
See also shared library routines
Dynamic widgets *299*
DYNAMIC-CAST function *546*
DYNAMIC-CURRENT-VALUE function *548*
DYNAMIC-CURRENT-VALUE statement *549*
DYNAMIC-ENUM function *550*
DYNAMIC-FUNCTION function *551*
DYNAMIC-INVOKE function *553*
DYNAMIC-NEW statement *555*
DYNAMIC-NEXT-VALUE function *558*
DYNAMIC-PROPERTY function *559*
DYNAMIC-PROPERTY statement *560*

E

EACH option
FOR statement *620*
OPEN QUERY statement *857*
PRESELECT phrase *914*
ECHO option
INPUT FROM statement *727*
INPUT THROUGH statement *733*
INPUT-OUTPUT THROUGH statement *738*
OUTPUT THROUGH statement *876*
OUTPUT TO statement *880*
EDGE-CHARS attribute *1510*
EDGE-CHARS option
DEFINE RECTANGLE statement *465*
EDGE-PIXELS attribute *1510*
EDGE-PIXELS option
DEFINE RECTANGLE statement *465*
EDIT-CAN-PASTE attribute *1510*
EDIT-CAN-UNDO attribute *1511*
EDIT-CLEAR method *1511*
EDIT-COPY method *1511*
EDIT-CUT method *1512*
EDIT-PASTE method *1512*
EDIT-UNDO method *1512*
EDITING phrase
PROMPT-FOR statement *929*
SET statement *1064*
UPDATE statement *1157*
EDITOR phrase
VIEW-AS phrase *1180*
EDITOR widget
attributes *1227*
dynamic *299*
events *1227*
methods *1227*
Elapsed time *592*
ELSE function *712*
ELSE statement *713*
EmbeddedWindow property *2105*
EMPTY attribute *1513*
EMPTY fill mode *1540*
EMPTY TEMP-TABLE statement *567*
EMPTY-DATASET method *1513*
EMPTY-SELECTION event *1954*
EMPTY-TEMP-TABLE method *1513*
Emumeration names *1140*

ENABLE method *1514*
ENABLE statement *568*
ENABLE-CONNECTIONS method *1514*
ENABLE-EVENTS method *1516*
ENABLED attribute *1516*
ENABLED-FIELDS option
  VALIDATE method *1887*
ENCODE function *573*
ENCODE-DOMAIN-ACCESS-CODE method *1517*
ENCODING attribute *1517*
ENCRYPT function *574*
ENCRYPT-AUDIT-MAC-KEY method *1520*
ENCRYPTION-SALT attribute *1521*
END CASE option
  CASE statement *161*
END CLASS option
  CLASS statement *185*
END COMPARES option
  BUFFER-COMPARE statement *144*
END CONSTRUCTOR option
  CONSTRUCTOR statement *260*
END DESTRUCTOR option
  DESTRUCTOR statement *523*
END INTERFACE option
  INTERFACE statement *747*
END METHOD option
  METHOD statement *671*, *808*, *919*
END option
  SEEK statement *1059*
END statement
  terminating a DO block *537*
  terminating a FOR block *620*
  terminating a REPEAT block *1001*
END_OF_ARRAY property *2109*
END-BOX-SELECTION event *1954*
END-DOCUMENT method *1521*
END-ELEMENT method *1522*
END-ERROR event *1941*
END-EVENT-GROUP method *1523*
END-FILE-DROP method *1523*
END-MOVE event *1952*
END-RESIZE event *1952*
END-ROW-RESIZE event *1952*
END-SEARCH event *1947*
END-USER-PROMPT attribute *1524*
ENDKEY event *1941*
ENTERED function *576*
ENTITY-EXPANSION-LIMIT attribute *1524*
ENTRY event *1948*
ENTRY function *577*
ENTRY method *1524*
ENTRY statement *579*
ENTRY-TYPES-LIST attribute *1525*
ENUM statement *581*
Enumeration member
  accessing *589*
Enumeration member access *589*
Enumerations
  built-in *1961*
  Progress.ApplicationServer.AdapterType *1963*
  Progress.Reflect.AccessMode *2019*
  Progress.Reflect.DataType *2021*
  Progress.Reflect.Flags *2022*
  Progress.Reflect.ParameterMode *2025*
EQ operator *590*
Equals( ) method *2109*
ERROR attribute *1525*
ERROR conditions
  RUN statement *1023*
  user-defined functions raising *671*
ERROR event *1941*
ERROR function *591*
ERROR option
  ON ENDKEY phrase *841*
  ON ERROR phrase *843*
  ON QUIT phrase *846*
  ON STOP phrase *854*
  RETURN statement *1012*
  UNDO statement *1149*
ERROR-COLUMN attribute *1526*
ERROR-OBJECT attribute *1526*
ERROR-OBJECT-DETAIL attribute *1527*
ERROR-ROW attribute *1527*
ERROR-STACK-TRACE attribute *1528*
ERROR-STATUS system handle
  attributes *1299*
  methods *1299*
ERROR-STRING attribute *1528*
ETIME function *592*
EVENT PROCEDURE option
  RUN statement *1023*
Event procedures (ActiveX)
  PROCEDURE statement *919*
Event procedures (asynchronous)
  PROCEDURE statement *919*
Event tables
  described *1940*
EVENT-GROUP-ID attribute *1529*
EVENT-HANDLER attribute *1529*
EVENT-HANDLER-OBJECT attribute *1529*
EVENT-PROCEDURE attribute *1530*
EVENT-PROCEDURE-CONTEXT attribute *1530*
EVENT-TYPE attribute *1531*
Events
  applying *119*, *1938*
  asynchronous request handle
    terminating with *1186*
  batch mode *1186*
  class *2272*
  database
    in schema triggers *1133*
    responding to *848*

Events *(continued)*
- DDE-NOTIFY *1236*
- default keyboard *1943*
- developer *1954*
- direct manipulation *1951*
- editing key function *1943*
- .NET *2272*
- frame-only direct manipulation *1953*
- general direct manipulation *1952*
- high-level widget *1946*
- keyboard *1940*
- low-level keyboard *1939*
- main key function classes *1940–1941*, *1943*
- MENU-DROP handling *848*
- mouse *1944*
- navigation key function *1941*
- priority *1938*
- procedure handle
  - responding to *848*
  - terminating with *1186*
- ProDataSet
  - FIND-FAILED *1959*
  - first-level FILL *1956*
  - OFF-END *1958*
  - row-level *1957*
  - second-level FILL *1957*
  - SYNCHRONIZE *1959*
- responding to
  - in schema triggers *1131*, *1133*
  - in user interface triggers *848*, *1131*
- socket *1955*
- universal key function *1940*
- user interface
  - responding to *848*, *1131*
- validation *1171*
- waiting for termination (.NET) *1190*
- waiting for termination (ABL) *1186*

Events in classes
- Progress.Data.BindingSource class *1968*
- Progress.Data.CancelCreateRowEventArgs class *1969*
- Progress.Data.ColumnPropDesc class *1971*
- Progress.Data.CreateRowEventArgs class *1972*
- Progress.Data.DummyRow class *1974*
- Progress.Data.InputValue class *1975*
- Progress.Data.OffEndEventArgs class *1976*
- Progress.Data.SortRequestEventArgs class *1977*
- Progress.Data.TableDesc class *1978*
- Progress.Database.TempTableInfo class *1979*
- Progress.Database.VSTTableId class *1980*
- Progress.IO.BinarySerializer class *1981*
- Progress.IO.FileInputStream class *1982*
- Progress.IO.FileOutputStream class *1983*
- Progress.IO.InputStream class *1984*
- Progress.IO.JsonSerializer class *1985*
- Progress.IO.MemoryInputStream class *1986*

Events in classes *(continued)*
- Progress.IO.MemoryOutputStream class *1989*
- Progress.IO.OutputStream class *1992*
- Progress.Json.JsonError class *1992*
- Progress.Json.JsonParser class *1993*
- Progress.Json.JsonParserError class *1994*
- Progress.Json.ObjectModel.JsonArray class *1994*
- Progress.Json.ObjectModel.JsonConstruct class *1996*
- Progress.Json.ObjectModel.JsonDataType class *1997*
- Progress.Json.ObjectModel.JsonObject class *1998*
- Progress.Json.ObjectModel.ObjectModelParser class *1999*
- Progress.Lang.AppError class *2000*
- Progress.Lang.Class class *2002*
- Progress.Lang.Enum class *2005*
- Progress.Lang.LockConflict class *2007*
- Progress.Lang.Object class *2008*
- Progress.Lang.OERequestInfo class *2009*
- Progress.Lang.OEVersionInfo class *2010*
- Progress.Lang.ParameterList class *2011*
- Progress.Lang.ProError class *2012*
- Progress.Lang.SoapFaultError class *2013*
- Progress.Lang.Stop class *2014*
- Progress.Lang.StopAfter class *2015*
- Progress.Lang.StopError class *2016*
- Progress.Lang.SysError class *2017*, *2027*
- Progress.Lang.UserInterrupt class *2018*
- Progress.Reflect.Constructor class *2020*
- Progress.Reflect.Event class *2022*
- Progress.Reflect.Method class *2024*
- Progress.Reflect.Parameter class *2025*
- Progress.Reflect.Property class *2026*
- Progress.Reflect.Variable class *2026*
- Progress.Util.DateTimeHelper class *2028*
- Progress.Util.DecimalHelper class *2029*
- Progress.Util.EnumHelper class *2030*
- Progress.Util.ResourceHelper class *2031*
- Progress.Util.StringHelper class *2032*
- Progress.Util.TypeHelper class *2033*
- Progress.Windows.Form class *2034*
- Progress.Windows.FormProxy class *2038*
- Progress.Windows.MDIChildForm class *2040*
- Progress.Windows.UserControl class *2045*
- Progress.Windows.WindowContainer class *2048*
- System.Exception class *2049*

Events on handles
- Asynchronous request object handle *1259*
- Buffer object handle *1263*
- ProDataSet object handle *1311*
- Server socket object handle *1326*
- Socket object handle *1336*

Events on widgets
- BROWSE widget *1208*
- BUTTON widget *1218*

Events on widgets *(continued)*
COMBO-BOX widget *1220*
CONTROL-FRAME widget *1222*
DIALOG-BOX widget *1225*
EDITOR widget *1227*
FIELD-GROUP widget *1230*
FILL-IN widget *1231*
FRAME widget *1233*
IMAGE widget *1237*
LITERAL widget *1238*
MENU widget *1239*
MENU-ITEM widget *1240*
RADIO-SET widget *1241*
RECTANGLE widget *1243*
SELECTION-LIST widget *1244*
SHADOW-WINDOW widget *1246*
SLIDER widget *1247*
SUB-MENU widget *1249*
TEXT widget *1250*
TOGGLE-BOX widget *1251*
WINDOW widget *1253*
Examples
index selections *620*
EXCEPT option
ASSIGN statement *126*
BUFFER-COMPARE statement *144*
BUFFER-COPY statement *146*
DEFINE BROWSE statement *366*
DEFINE FRAME statement *410*
DEFINE QUERY statement *460*
DISABLE statement *525*
DISPLAY statement *531*
ENABLE statement *568*
EXPORT statement *594*
FORM statement *632*
IMPORT statement *717*
INSERT statement *742*
PROMPT-FOR statement *929*
Record phrase *985*
SET statement *1064*
UPDATE statement *1157*
EXCLUSIVE-ID attribute *1531*
EXCLUSIVE-LOCK option
DEFINE BROWSE statement *366*
FIND statement *609*
FIND-BY-ROWID method *1541*
GET statement *682*
GET-CURRENT method *1570*
GET-FIRST method *1574*
GET-LAST method *1577*
GET-NEXT method *1580*
GET-PREV method *1582*
Record phrase *985*
EXECUTION-LOG attribute *1531*
EXIT-CODE attribute *1531*
EXP function *593*
EXPAND attribute *1532*
EXPAND option
RADIO-SET phrase *975*
EXPANDABLE attribute *1533*
EXPANDABLE option
DEFINE BROWSE statement *366*
EXPLICIT COMPARES option
BUFFER-COMPARE statement *144*
EXPORT method *1533*
EXPORT option
frame phrase *647*
EXPORT statement
blb files *594*
EXPORT-PRINCIPAL method *1534*
Expression *597*
Expressions
precedence *66*
EXTENDED option
QUERY-TUNING phrase *967*
EXTENT attribute *1535*
EXTENT function *602*
EXTENT option
DEFINE PROPERTY statement *442*, *671*
DEFINE VARIABLE statement *429*, *490*
METHOD statement *671*, *808*
Parameter definition syntax *892*
Extent property *2110*
EXTENT statement *603*
EXTERNAL option
PROCEDURE statement *919*
External procedures
running *1023*

F

FETCH-SELECTED-ROW method *1535*
FGCOLOR attribute *1535*
FGCOLOR option
DEFINE BROWSE statement *366*
DEFINE BUTTON statement *386*
DEFINE FRAME statement *410*
DEFINE IMAGE statement *419*
DEFINE MENU statement *423*
DEFINE RECTANGLE statement *465*
DEFINE SUB-MENU statement *471*
DEFINE VARIABLE statement *490*
ENABLE statement *568*
FORM statement *632*
format phrase *638*
frame phrase *647*
PROMPT-FOR statement *929*
Field editing key function events *1943*
Field groups
background *632*
BACKGROUND option
DEFINE FRAME statement *410*

Field groups *(continued)*
data *410*, *632*
header *632*
HEADER option
DEFINE FRAME statement *410*
Field lists
DEFINE QUERY statement *460*
Record phrase *985*
FIELD option
CHOOSE statement *168*
DEFINE TEMP-TABLE statement *477*
DEFINE WORK-TABLE statement *504*
RAW-TRANSFER statement *980*
Widget phrase *1201*
Field options
DEFINE TEMP-TABLE statement *477*
DEFINE WORK-TABLE statement *504*
FIELD-GROUP widget
attributes *1230*
events *1230*
methods *1230*
FieldIndex property *2110*
FieldName property *2110*
FIELDS option
DEFINE QUERY statement *460*
Record phrase *985*
FILE option
COPY-LOB statement *264*
Image phrase *715*
FILE-CREATE-DATE attribute *1536*
FILE-CREATE-TIME attribute *1536*
FILE-INFO system handle
attributes *1302*
FILE-MOD-DATE attribute *1536*
FILE-MOD-TIME attribute *1537*
FILE-NAME attribute *1537*
FILE-NAME preprocessor name *74*
FILE-OFFSET attribute *1538*
FILE-SIZE attribute *1538*
FILE-TYPE attribute *1538*
FileListId property *2111*
FileName property *2111*
FILL events *1956*
FILL function *605*
FILL method *1539*
FILL-IN option
VIEW-AS phrase *1180*
FILL-IN widget
attributes *1231*
dynamic *299*
events *1231*
methods *1231*
FILL-MODE attribute *1540*
FILL-WHERE-STRING attribute *1541*
FILLED attribute *1540*
FILTERS option
SYSTEM-DIALOG GET-FILE statement *1110*
FINAL option
CLASS statement *188*
METHOD statement *808*
Finally block
FOR statement *620*
FINALLY statement *606*
FIND statement
unique *609*
FIND-BY-ROWID method *1541*
FIND-CURRENT method *1542*
FIND-FAILED event *1959*
FIND-FIRST method *1543*
FIND-LAST method *1545*
FIND-UNIQUE method *1546*
FINDER option
SYSTEM-HELP statement *1115*
FIRST function *618*
FIRST option
CAN-FIND function *155*
FIND statement *609*
FOR statement *620*
GET statement *682*
OPEN QUERY statement *857*
PRESELECT phrase *914*
FIRST-ASYNC-REQUEST attribute *1547*
FIRST-BUFFER attribute *1547*
FIRST-CHILD attribute *1548*
FIRST-COLUMN attribute *1548*
FIRST-DATA-SOURCE attribute *1548*
FIRST-DATASET attribute *1548*
FIRST-FORM attribute *1549*
First-level FILL events *1956*
FIRST-OBJECT attribute *1549*
FIRST-OF function *619*
FIRST-OF method *1550*
FIRST-PROCEDURE attribute *1551*
FIRST-QUERY attribute *1551*
FIRST-SERVER attribute *1551*
FIRST-SERVER-SOCKET attribute *1552*
FIRST-SOCKET attribute *1552*
FIRST-TAB-ITEM attribute *1552*
FIT-LAST-COLUMN attribute *1552*
FIT-LAST-COLUMN option
DEFINE BROWSE statement *366*
FIX-CODEPAGE statement *619*
FIXED-ONLY option
SYSTEM-DIALOG FONT statement *1107*
FLAGS option
ENUM statement *581*
FLAT-BUTTON attribute *1554*
FLAT-BUTTON option
DEFINE BUTTON statement *386*
Flush( ) method (FileOutputStream) *2111*
Flush( ) method (OutputStream) *2112*

Focus
ACTIVE-WINDOW system handle 1258
CANCEL-BUTTON attribute 1422
CHOOSE event 1947
DEFAULT attribute 1492
DEFAULT option
DEFINE BUTTON statement 386
DEFAULT-BUTTON attribute 1493
ENABLE statement 568
ENTRY event 1948
FIRST-TAB-ITEM attribute 1552
FOCUS system handle 1304
Frame phrase
CANCEL-BUTTON option 647
DEFAULT-BUTTON option 647
FRAME-FIELD function 663
LAST-TAB-ITEM attribute 1645
LEAVE event 1948
navigation key function events 1941
portable mouse events 1944
SENSITIVE attribute 1806
three-button mouse events 1945
UPDATE statement 1157
WAIT-FOR statement
FOCUS Option 1186
WIDGET-ENTER attribute 1897
WIDGET-LEAVE attribute 1898
FOCUS option
WAIT-FOR statement 1186
FOCUS system handle
attributes 1304
FOCUSED-ROW attribute 1554
FOCUSED-ROW-SELECTED attribute 1554
FONT attribute 1554
FONT option
DEFINE BROWSE statement 366
DEFINE BUTTON statement 386
DEFINE FRAME statement 410
DEFINE MENU statement 423
DEFINE SUB-MENU statement 471
DEFINE VARIABLE statement 490
ENABLE statement 568
FORM statement 632
format phrase 638
frame phrase 647
PROMPT-FOR statement 929
PUT-KEY-VALUE statement 958
FONT-TABLE system handle
attributes 1305
methods 1305
FOR DATABASE option
CREATE ALIAS statement 271
FOR option
COPY-LOB statement 264
DEFINE BUFFER statement 380
DEFINE DATA-SOURCE statement 401
FOR option *(continued)*
DEFINE DATASET statement 393
DEFINE QUERY statement 460
DO statement 537
OPEN QUERY statement 857
REPEAT statement 1001
FOR statement 620
FOR TABLE option
CREATE BUFFER statement 281
FORCE-FILE option
SYSTEM-HELP statement 1115
FOREGROUND attribute 1555
FOREIGN-KEY-HIDDEN attribute 1555
Form item
DEFINE FRAME statement 410
FORM statement 632
FORM-INPUT attribute 1556
FORM-LONG-INPUT attribute 1556
FORMAT attribute 1556
FORMAT option
DEFINE PARAMETER statement 429
DEFINE VARIABLE statement 490
format phrase 638
MESSAGE statement 799
PUT statement 948
Format phrase
DEFINE FRAME statement 410
determining labels 638
DISPLAY statement 531
ENABLE statement 568
FORM statement 632
PROMPT-FOR statement 929
UPDATE statement 1157
FORMAT phrase
SET statement 1064
FORMATTED attribute 1556
Formatted property 2112
FORWARD option
FUNCTION statement 671
REPOSITION statement 1008
FORWARD-ONLY attribute 1557
FRAGMENT attribute 1557
FRAME attribute 1558
Frame families 1233
FRAME option
ASSIGN statement 126
CAN-FIND function 155
CLEAR statement 204
DDE INITIATE statement 358
DEFINE FRAME statement 410
ENTERED function 576
FIND statement 609
frame phrase 647
INPUT function 724
NOT ENTERED function 834
Record phrase 985

FRAME option *(continued)*
Widget phrase *1201*
Frame phrase
CHOOSE statement *168*
COLOR statement *213*
DISABLE statement *525*
DISPLAY statement *531*
DO statement *537*, *671*, *808*, *919*
DOWN statement *544*
ENABLE statement *568*
FOR statement *620*
FORM statement *632*
INSERT statement *742*
NEXT-PROMPT statement *828*
PROMPT-FOR statement *929*
REPEAT statement *1001*
SCROLL statement *1049*
SET statement *1064*
UNDERLINE statement *1148*
UP statement *1156*
UPDATE statement *1157*
FRAME widget
attributes *1233*
dynamic *299*
events *1233*
methods *1233*
FRAME-COL attribute *1558*
FRAME-COL function *660*
FRAME-DB function *661*
FRAME-DOWN function *662*
FRAME-FIELD function *663*
FRAME-FILE function *664*
FRAME-INDEX function *664*
FRAME-LINE function *665*
FRAME-NAME attribute *1558*
FRAME-NAME function *667*
FRAME-ROW attribute *1558*
FRAME-ROW function *668*
FRAME-SPACING attribute *1559*
FRAME-VALUE function *669*
FRAME-VALUE statement *670*
FRAME-X attribute *1559*
FRAME-Y attribute *1559*
FREQUENCY attribute *1560*
FREQUENCY option
SLIDER phrase *1085*
FROM option
CREATE DATABASE statement *285*
Image phrase *715*
PUBLISH statement *941*
FROM-CURRENT option
SCROLL statement *1049*
FULL-HEIGHT-CHARS attribute *1560*
FULL-HEIGHT-PIXELS attribute *1560*
FULL-PATHNAME attribute *1560*
FULL-WIDTH-CHARS attribute *1561*
FULL-WIDTH-PIXELS attribute *1561*
FUNCTION attribute *1561*
Function body
FUNCTION statement *671*
raising ERROR conditions *671*
FUNCTION statement *671*
Functions
user-defined *671*

G

GATEWAYS function *678*
GE operator *678*
GENERATE-PBE-KEY function *679*
GENERATE-PBE-SALT function *680*
GENERATE-RANDOM-KEY function *681*
GENERATE-UUID function *681*
Generic types, .NET *317*
GET option
DEFINE PROPERTY statement *442*
GET statement *682*
GET-ATTRIBUTE method *1561*
GET-ATTRIBUTE-NODE method *1562*
GET-BINARY-DATA method *1563*
GET-BITS function *684*
GET-BLUE-VALUE method *1563*
GET-BROWSE-COLUMN method *1563*
GET-BUFFER-HANDLE method *1564*
GET-BYTE function *685*
GET-BYTE-ORDER function *686*
GET-BYTES function *686*
GET-BYTES-AVAILABLE method *1564*
GET-CALLBACK-PROC-CONTEXT method *1565*
GET-CALLBACK-PROC-NAME method *1565*
GET-CGI-LIST method *1566*
GET-CGI-LONG-VALUE method *1566*
GET-CGI-VALUE method *1566*
GET-CHANGES method *1566*
GET-CHILD method *1567*
GET-CHILD-RELATION method *1568*
GET-CLASS function *687*
GET-CLIENT method *1568*
GET-CODEPAGE function *688*
GET-CODEPAGES function *688*
GET-COLLATION function *689*
GET-COLLATIONS function *689*
GET-COLUMN method *1569*
GET-CONFIG-VALUE method *1569*
GET-CURRENT method *1570*
GET-DATASET-BUFFER method *1570*
GET-DB-CLIENT function *690*
GET-DOCUMENT-ELEMENT method *1571*
GET-DOUBLE function *692*
GET-DROPPED-FILE method *1571*
GET-DYNAMIC method *1572*
GET-EFFECTIVE-TENANT-ID function *693*

GET-EFFECTIVE-TENANT-NAME function *693*
GET-ERROR-COLUMN method *1572*
GET-ERROR-ROW method *1573*
GET-FILE-NAME method *1573*
GET-FILE-OFFSET method *1573*
GET-FIRST method *1574*
GET-FLOAT function *694*
GET-GREEN-VALUE method *1575*
GET-HEADER-ENTRY method *1575*
GET-INDEX-BY-NAMESPACE-NAME method *1575*
GET-INDEX-BY-QNAME method *1576*
GET-INT64 function *695*
GET-ITERATION method (Data Objects) *1576*
GET-ITERATION method (Widget Objects) *1577*
GET-KEY-VALUE statement *696*
GET-LAST method *1577*
GET-LOCALNAME-BY-INDEX method *1578*
GET-LONG function *699*
GET-MESSAGE method (Handle) *1579*
GET-MESSAGE-TYPE method *1579*
GET-NEXT method *1580*
GET-NODE method *1580*
GET-NUMBER method *1581*
GET-PARENT method *1581*
GET-POINTER-VALUE function *699*
GET-PREV method *1582*
GET-PRINTERS method *1583*
GET-PROPERTY method *1583*
GET-QNAME-BY-INDEX method *1584*
GET-RED-VALUE method *1584*
GET-RELATION method *1585*
GET-REPOSITIONED-ROW method *1585*
GET-RGB-VALUE method *1585*
GET-ROW method *1586*
GET-SAFE-USER method *1586*
GET-SELECTED-WIDGET method *1587*
GET-SERIALIZED method *1587*
GET-SHORT function *701*
GET-SIGNATURE method *1588*
GET-SIZE function *701*
GET-SOCKET-OPTION method *1590*
GET-SOURCE-BUFFER method *1591*
GET-STRING function *703*
GET-TAB-ITEM method *1591*
GET-TEXT-HEIGHT-CHARS method *1592*
GET-TEXT-HEIGHT-PIXELS method *1592*
GET-TEXT-WIDTH-CHARS method *1593*
GET-TEXT-WIDTH-PIXELS method *1593*
GET-TOP-BUFFER method *1594*
GET-TYPE-BY-INDEX method *1594*
GET-TYPE-BY-NAMESPACE-NAME method *1595*
GET-TYPE-BY-QNAME method *1595*
GET-UNSIGNED-LONG function *703*
GET-UNSIGNED-SHORT function *704*
GET-URI-BY-INDEX method *1596*
GET-VALUE-BY-INDEX method *1596*
GET-VALUE-BY-NAMESPACE-NAME *1597*
GET-VALUE-BY-QNAME method *1597*
GET-WAIT-STATE method *1597*
Get( ) method *2112*
GetCharacter( ) method (JsonArray) *2113*
GetCharacter( ) method (JsonObject) *2114*
GetClass( ) method (Progress.Lang.Class) *2115*
GetClass( ) method (Progress.Lang.Object) *2115*
GetClientPrincipal( ) method (Progress.Lang.OERequestInfo) *2116*
GetCOMHandle( ) method (JsonArray) *2117*
GetCOMHandle( ) method (JsonObject) *2117*
GetConstructor( ) method *2118*
GetConstructors( ) method *2119*
GetDate( ) method (JsonArray) *2120*
GetDate( ) method (JsonObject) *2120*
GetDatetime( ) method (JsonArray) *2121*
GetDatetime( ) method (JsonObject) *2122*
GetDatetimeTZ( ) method (JsonArray) *2122*
GetDatetimeTZ( ) method (JsonObject) *2123*
GetDecimal( ) method (JsonArray) *2124*
GetDecimal( ) method (JsonObject) *2125*
GetEnum( ) method *2125*
GetEnumName( ) method *2127*
GetEnumNames( ) method *2127*
GetEnumValue( ) method *2128*
GetEnumValues( ) method *2128*
GetEvent( ) method *2129*
GetEvents( ) method *2129*
GetHandle( ) method (JsonArray) *2130*
GetHandle( ) method (JsonObject) *2131*
GetHandlerParameters( ) method *2131*
GetIndexInfoByID( ) method *2132*
GetIndexStatHistoryHandle() method *2133*
GetInt64( ) method (JsonArray) *2135*
GetInt64( ) method (JsonObject) *2136*
GetInteger( ) method (JsonArray) *2133*
GetInteger( ) method (JsonObject) *2134*
GetInterfaces( ) method *2135*
GetJsonArray( ) method (JsonArray) *2136*
GetJsonArray( ) method (JsonObject) *2138*
GetJsonObject( ) method (JsonArray) *2139*
GetJsonObject( ) method (JsonObject) *2140*
GetJsonText( ) method (JsonArray) *2142*
GetJsonText( ) method (JsonObject) *2143*
GetLogical( ) method (JsonArray) *2143*
GetLogical( ) method (JsonObject) *2144*
GetLongchar( ) method (JsonArray) *2145*
GetLongchar( ) method (JsonObject) *2146*
GetMemptr( ) method (JsonArray) *2146*
GetMemptr( ) method (JsonObject) *2147*
GetMessage( ) method (Class) *2148*
GetMessageNum( ) method *2148*
GetMethod( ) method *2149*
GetMethods( ) method *2150*
GetNames( ) method (JsonObject) *2151*

GetParameters( ) method *2151*
GetProcessTemplateList( ) method *2155*
GetProperties( ) method *2152*
GetProperty( ) method *2152*
GetPropertyValue( ) method *2153*
GetRaw( ) method (JsonArray) *2155*
GetRaw( ) method (JsonObject) *2156*
GetRecid( ) method (JsonArray) *2156*
GetRecid( ) method (JsonObject) *2157*
GetRowid( ) method (JsonArray) *2157*
GetRowid( ) method (JsonObject) *2158*
GetTableInfoByID( ) method *2159*
GetTableInfoByPosition( ) method *2160*
GetTableStatHistoryHandle( ) method *2161*
GetterAccessMode property *2162*
GetType( ) method (JsonArray) *2162*
GetType( ) method (JsonObject) *2162*
GetType( ) method (TypeHelper) *2163*
GetValue( ) method *2165*
GetVariable( ) method *2164*
GetVariables( ) method *2164*
GetVSTHandle( ) method *2166*
GO event *1941*
GO-ON option
  CHOOSE statement *168*
  PROMPT-FOR statement *929*
  SET statement *1064*
  UPDATE statement *1157*
GO-PENDING function *705*
GRAPHIC-EDGE attribute *1598*
GRAPHIC-EDGE option
  DEFINE RECTANGLE statement *465*
Greater Than operator *706*
Greater Than or Equal to operator *678*
GRID-FACTOR-HORIZONTAL attribute *1598*
GRID-FACTOR-VERTICAL attribute *1598*
GRID-SNAP attribute *1599*
GRID-UNIT-HEIGHT-CHARS attribute *1599*
GRID-UNIT-HEIGHT-PIXELS attribute *1599*
GRID-UNIT-WIDTH-CHARS attribute *1600*
GRID-UNIT-WIDTH-PIXELS attribute *1600*
GRID-VISIBLE attribute *1600*
GROUP-BOX attribute *1601*
GROUP-BOX option
  DEFINE RECTANGLE statement *465*
GT operator *706*
GUID function *707*

H

HANDLE attribute *1601*
HANDLE data type *318*
HANDLE function *708*
HANDLE option *429*
Handle property *2166*
Handle-based objects
  accessing
    attributes *1356*
    methods *1357*
  attributes and methods *1353*
  defined *315*
  Reference *1257*
  referencing
    by handle *1356*
    by name *1355*
    chained *1357*
HANDLER attribute *1602*
HandleRequest( ) method (IWebHandler) *2167*
Handles
  functions
    HANDLE function *708*
    VALID-HANDLE function *1171*
    WIDGET-HANDLE function *1200*
  procedure *1023*
  validating *1171*, *1173*
HAS-LOBS attribute *1603*
HAS-RECORDS attribute *1603*
Has( ) method (JsonObject) *2167*
HasStatics( ) method *2168*
HasWidgetPool( ) method *2168*
Head item
  DEFINE FRAME statement *410*
HEADER option
  DEFINE FRAME statement *410*
  FORM statement *632*
Height property *1603*
HEIGHT-CHARS attribute *1604*
HEIGHT-PIXELS attribute *1604*
HELP attribute *1604*
HELP event *1941*
HELP option
  CHOOSE statement *168*
  DEFINE BROWSE statement *366*
  DEFINE TEMP-TABLE statement *477*
  format phrase *638*
  SYSTEM-HELP statement *1115*
HELP-TOPIC option
  SYSTEM-HELP statement *1115*
HEX-DECODE function *709*
HEX-ENCODE function *709*
HIDDEN attribute *1605*
HIDE statement *710*
High-level widget events *1946*
HINT option
  QUERY-TUNING phrase *967*
HonorProKeys property *1606*
HonorReturnKey property *1606*
HORIZONTAL attribute *1607*
HORIZONTAL option
  RADIO-SET phrase *975*
  SLIDER phrase *1085*

HTML-CHARSET attribute *1607*
HTML-END-OF-LINE attribute *1607*
HTML-END-OF-PAGE attribute *1608*
HTML-FRAME-BEGIN attribute *1608*
HTML-FRAME-END attribute *1608*
HTML-HEADER-BEGIN attribute *1608*
HTML-HEADER-END attribute *1609*
HTML-TITLE-BEGIN attribute *1609*
HTML-TITLE-END attribute *1609*
HWND attribute *1609*

I

ICFPARAMETER attribute *1610*
ICON attribute *1610*
ID list
  values *126*, *152*, *266*
IF function *712*
IF statement *713*
IF... THEN... ELSE function *712*
IF... THEN... ELSE statement *713*
IGNORE-CURRENT-MODIFIED attribute *1610*
IgnoreComments property *2168*
IMAGE attribute *1610*
Image file formats *715*
IMAGE option
  DEFINE BUTTON statement *386*
  DEFINE IMAGE statement *419*
Image phrase
  DEFINE IMAGE statement *419*
IMAGE widget
  attributes *1237*
  dynamic *299*
  events *1237*
  methods *1237*
IMAGE-DOWN attribute *1611*
IMAGE-DOWN option
  DEFINE BUTTON statement *386*
IMAGE-INSENSITIVE attribute *1611*
IMAGE-INSENSITIVE option
  DEFINE BUTTON statement *386*
IMAGE-SIZE option
  Image phrase *715*
IMAGE-SIZE-CHARS option
  Image phrase *715*
IMAGE-SIZE-PIXELS option
  Image phrase *715*
IMAGE-UP attribute *1611*
IMAGE-UP option
  DEFINE BUTTON statement *386*
Images *715*
IMMEDIATE-DISPLAY attribute *1611*
IMPLEMENTS option
  CLASS statement *187*
IMPORT statement *717*
IMPORT-NODE method *1612*
IMPORT-PRINCIPAL method *1612*
IN BROWSE option
  Widget phrase *1201*
IN FRAME option
  Widget phrase *1201*
IN option
  DYNAMIC-FUNCTION function *551*
  FUNCTION statement *671*
  RUN statement
    internal procedure calls *1023*
  SUBSCRIBE statement *1096*
  UNSUBSCRIBE statement *1155*
IN SUPER option
  FUNCTION statement *671*
  PROCEDURE statement *919*
IN WIDGET-POOL option
  CREATE BROWSE statement *278*
  CREATE CALL statement *283*
  CREATE CLIENT-PRINCIPAL statement *284*
  CREATE DATA-SOURCE statement *288*
  CREATE DATASET statement *287*
  CREATE QUERY statement *289*
  CREATE SAX-ATTRIBUTES statement *290*
  CREATE SAX-READER statement *293*
  CREATE SAX-WRITER statement *294*
  CREATE SOAP-HEADER statement *295–296*
  CREATE TEMP-TABLE statement *297*
  CREATE widget statement *299*
  CREATE X-DOCUMENT statement *304*
  CREATE X-NODEREF statement *305*
  CREATE-BUFFER statement *281*
IN WINDOW option
  DISPLAY statement *531*
  ENABLE statement *568*
  frame phrase *647*
  HIDE statement *710*
  MESSAGE statement *799*
  PAUSE statement *912*
  PROMPT-FOR statement *929*
  STATUS statement *1089*
  SYSTEM-DIALOG COLOR statement *1106*
  SYSTEM-DIALOG FONT statement *1107*
  SYSTEM-DIALOG GET-FILE statement *1110*
  SYSTEM-DIALOG PRINTER-SETUP statement *1114*
  VIEW statement *1178*
IN-HANDLE attribute *1615*
IN-MENU option
  Widget phrase *1201*
Include file reference *70*
INCREMENT-EXCLUSIVE-ID method *1613*
Indeterminate arrays *317*
INDEX attribute *1613*
INDEX function *722*
INDEX option
  DEFINE TEMP-TABLE statement *477*

Index selections
examples *620*
INDEX-HINT option
QUERY-TUNING phrase *967*
INDEX-INFORMATION attribute *1614*
INDEX-INFORMATION method *1615*
INDEXED-REPOSITION option
OPEN QUERY statement *857*
IndexStatId property *2169*
INHERIT-BGCOLOR attribute *1617*
INHERIT-BGCOLOR option
frame phrase *647*
INHERIT-FGCOLOR attribute *1617*
INHERIT-FGCOLOR option
frame phrase *647*
INHERITS option
CLASS statement *186*
INTERFACE statement *747*
INITIAL attribute *1618*
INITIAL option
DEFINE PARAMETER statement *429*
DEFINE PROPERTY statement *442*
DEFINE VARIABLE statement *490*
INITIAL-DIR option
SYSTEM-DIALOG GET-DIR statement *1109*
SYSTEM-DIALOG GET-FILE statement *1110*
INITIAL-FILTER option
SYSTEM-DIALOG GET-FILE statement *1110*
INITIALIZE method *1619*
INITIALIZE-DOCUMENT-TYPE method *1620*
INITIATE method *1622*
INNER-CHARS attribute *1622*
INNER-CHARS option
EDITOR phrase *564*
SELECTION-LIST phrase *1061*
INNER-LINES attribute *1623*
INNER-LINES option
COMBO-BOX phrase *215*
EDITOR phrase *564*
SELECTION-LIST phrase *1061*
INPUT CLEAR statement *725*
INPUT CLOSE statement *726*
INPUT FROM statement *727*
INPUT function *724*
INPUT OFF option
STATUS statement *1089*
INPUT option
COLOR phrase *210*
Parameter definition syntax *892*
Parameter passing syntax *898*
SEEK function *1058*
SEEK statement *1059*
STATUS statement *1089*
stored procedures *1041*
INPUT PARAMETER option
DEFINE PARAMETER statement *429*
INPUT THROUGH statement *733*
INPUT-OUTPUT CLOSE statement *737*
INPUT-OUTPUT option
Parameter definition syntax *892*
Parameter passing syntax *898*
stored procedures *1041*
INPUT-OUTPUT PARAMETER option
DEFINE PARAMETER statement *429*
INPUT-OUTPUT THROUGH statement *738*
INPUT-VALUE attribute *1623*
InputValue property *2169*
INSERT method *1623*
INSERT statement *742*
INSERT-ATTRIBUTE method *1624*
INSERT-BACKTAB method *1627*
INSERT-BEFORE method *1627*
INSERT-FILE method *1628*
INSERT-ROW method *1628*
INSERT-STRING method *1628*
INSERT-TAB method *1629*
INSTANTIATING-PROCEDURE attribute *1629*
INT data type *318*
INT64 data type *318*
INT64 function *744*
INTEGER data type *318*
INTEGER function *746*
Interface body
INTERFACE statement *747*
Interface names *1140*
INTERFACE statement *747*
Interfaces
built-in *1961*
Progress.Lang.Error *2005*
Progress.Web.IWebHandler *2034*
Progress.Windows.IForm *2039*
user-defined *747*
Internal procedures
asynchronous event procedures *1023*
invoking automatic transactions *1130*
OCX event procedures *919*
running *1023*
shared library procedures *1023*
INTERNAL-ENTRIES attribute *1631*
INTERVAL function *757*
INTO option
COMPILE statement *222*
InvalidConfiguration property *2169*
INVOKE method (Handle) *1631*
Invoke( ) method (Class) *2170*
Invoke( ) method (Constructor) *2171*
Invoke( ) method (Method) *2172*
IS option
DEFINE TEMP-TABLE statement *477*
IS-ATTR-SPACE function *758*
IS-CLASS attribute *1633*
IS-CODEPAGE-FIXED function *758*

IS-COLUMN-CODEPAGE function *758*
IS-DB-MULTI-TENANT function *759*
IS-JSON attribute *1633*
IS-LEAD-BYTE function *759*
IS-MULTI-TENANT attribute *1633*
IS-OPEN attribute *1633*
IS-PARAMETER-SET attribute *1634*
IS-PARTITIONED attribute *1634*
IS-ROW-SELECTED method *1634*
IS-SELECTED method *1635*
IS-XML attribute *1635*
IsA( ) method *2172*
IsAbstract property *2174*
IsAbstract( ) method *2173*
IsEnum( ) method *2174*
IsFinal property *2175*, *2179*
IsFinal( ) method *2174*
IsFlagsEnum( ) method *2175*
IsFlagSet( ) method *2175*
IsGeneric( ) method *2177*
IsGreater( ) method *2177*
IsGreaterOrEqual( ) method *2178*
IsIndexed( ) method *2179*
IsInterface( ) method *2179*
IsLess( ) method *2179*
IsLessOrEqual( ) method *2180*
IsNull( ) method (JsonArray) *2181*
IsNull( ) method (JsonObject) *2182*
ISO-DATE function *760*
IsOverride property *2182*
IsSerializable( ) method *2182*
IsStatic property *2183*
ITEM option
  DDE ADVISE statement *352*
  DDE GET statement *356*
  DDE REQUEST statement *359*
  DDE SEND statement *361*
ITEMS-PER-ROW attribute *1635*
ITERATION-CHANGED event *1948*

J

JOIN-BY-SQLDB option
  QUERY-TUNING phrase *967*
Joins
  inner *985*
  left outer *985*

K

KBLABEL function *760*
KEEP-CONNECTION-OPEN attribute *1636*
KEEP-FRAME-Z-ORDER attribute *1636*
KEEP-MESSAGES option
  OUTPUT TO statement *880*
KEEP-SECURITY-CACHE attribute *1636*
KEEP-TAB-ORDER option
  frame phrase *647*
Key actions
  Text field *929*, *1064*, *1157*
KEY attribute *1637*
Key function events
  main classes *1940–1941*, *1943*
KEY option
  GET-KEY-VALUE statement *696*
  PUT-KEY-VALUE statement *958*
  SYSTEM-HELP statement *1115*
Keyboard Events *1940*
KEYCODE function *761*
KEYFUNCTION function *762*
KEYLABEL function *764*
Keys
  ON ENDKEY phrase *841*
  ON statement *848*
KEYS attribute *1637*
KEYS option
  CHOOSE statement *168*
  DEFINE DATA-SOURCE statement *401*
KEYWORD function *764*
KEYWORD-ALL function *766*
Keywords
  Index *2287*

L

L option
  Character-string literal (" ") *67*
LABEL attribute *1638*
LABEL option
  aggregate phrase *111*
  DEFINE BUFFER statement *380*
  DEFINE BUTTON statement *386*
  DEFINE MENU statement *423*
  DEFINE PARAMETER statement *429*
  DEFINE SUB-MENU statement *471*
  DEFINE VARIABLE statement *490*
  format phrase *638*
Label rules
  format phrase *638*
LABEL-BGCOLOR attribute *1638*
LABEL-BGCOLOR option
  DEFINE BROWSE statement *366*
LABEL-DCOLOR attribute *1639*
LABEL-DCOLOR option
  DEFINE BROWSE statement *366*
LABEL-FGCOLOR attribute *1639*
LABEL-FGCOLOR option
  DEFINE BROWSE statement *366*
LABEL-FONT attribute *1639*
LABEL-FONT option
  DEFINE BROWSE statement *366*
LABELS attribute *1639*

LABELS-HAVE-COLONS attribute *1640*
LANDSCAPE option
  OUTPUT TO statement *880*
  SYSTEM-DIALOG PRINTER-SETUP statement *1114*
LANGUAGES attribute *1640*
LANGUAGES option
  COMPILE statement *222*
LARGE attribute *1640*
LARGE option
  EDITOR phrase *564*
LARGE-TO-SMALL attribute *1640*
LARGE-TO-SMALL option
  SLIDER phrase *1085*
LAST function *767*
LAST keyword value
  DBRESTRICTIONS function *347*
LAST option
  CAN-FIND function *155*
  FIND statement *609*
  FOR statement *620*
  GET statement *682*
  OPEN QUERY statement *857*
  PRESELECT phrase *914*
LAST-ASYNC-REQUEST attribute *1641*
LAST-BATCH attribute *1641*
LAST-CHILD attribute *1641*
LAST-EVENT system handle
  attributes *1307*
LAST-FORM attribute *1642*
LAST-OBJECT attribute *1642*
LAST-OF function *769*
LAST-OF method *1643*
LAST-PROCEDURE attribute *1644*
LAST-SERVER attribute *1644*
LAST-SERVER-SOCKET attribute *1644*
LAST-SOCKET attribute *1645*
LAST-TAB-ITEM attribute *1645*
LASTKEY function *767*
LC function *769*
LDBNAME function *770*
LE operator *772*
LEAVE event *1948*
LEAVE option
  ON ENDKEY phrase *841*
  ON ERROR phrase *843*
  ON QUIT phrase *846*
  ON STOP phrase *854*
  UNDO statement *1149*
LEAVE statement *773*
LEFT event *1942*
LEFT OUTER-JOIN option
  Record phrase *985*
Left property *1645*
LEFT-ALIGNED option
  AT phrase *131*
LEFT-MOUSE-CLICK event *1946*
LEFT-MOUSE-DBLCLICK event *1946*
LEFT-MOUSE-DOWN event *1945*
LEFT-MOUSE-UP event *1946*
LEFT-TRIM function *774*
LENGTH attribute *1646*
LENGTH function *775*
Length property (IO) *2183*
Length property (JSON) *2183*
LENGTH statement *777*
Less Than operator *794*
Less Than or Equal to operator *772*
LIBRARY attribute *1646*
LIBRARY function *778*
LIBRARY-CALLING-CONVENTION attribute *1646*
LIKE option
  DEFINE BUTTON statement *386*
  DEFINE IMAGE statement *419*
  DEFINE MENU statement *423*
  DEFINE PARAMETER statement *429*
  DEFINE RECTANGLE statement *465*
  DEFINE SUB-MENU statement *471*
  DEFINE TEMP-TABLE statement *477*
  DEFINE VARIABLE statement *490*
  DEFINE WORK-TABLE statement *504*
  Format phrase *638*
  MESSAGE statement *799*
  Parameter definition syntax *892*
LIKE-SEQUENTIAL option
  DEFINE TEMP-TABLE statement *477*
LINE attribute *1647*
LINE-COUNTER function *779*
LINE-NUMBER preprocessor name *74*
LIST-EVENTS function *781*
LIST-ITEM-PAIRS attribute *1647*
LIST-ITEM-PAIRS option
  COMBO-BOX phrase *215*
  SELECTION-LIST phrase *1061*
LIST-ITEMS attribute *1648*
LIST-ITEMS option
  COMBO-BOX phrase *215*
  SELECTION-LIST phrase *1061*
LIST-PROPERTY-NAMES method *1648*
LIST-QUERY-ATTRS function *782*
LIST-SET-ATTRS function *783*
LIST-WIDGETS function *784*
LISTING option
  COMPILE statement *222*
LISTINGS attribute *1649*
LITERAL widget
  attributes *1238*
  events *1238*
  methods *1238*
LITERAL-QUESTION attribute *1649*
LOAD method (Handle) *1650*

LOAD option
DISABLE TRIGGERS statement 528
LOAD statement 785
LOAD-DOMAINS method 1652
LOAD-ICON method 1653
LOAD-IMAGE method 1654
LOAD-IMAGE-DOWN method 1655
LOAD-IMAGE-INSENSITIVE method 1656
LOAD-IMAGE-UP method 1657
LOAD-MOUSE-POINTER method 1658
LOAD-PICTURE statement 788
LOAD-RESULT-INTO option
stored procedures 1041
LOAD-SMALL-ICON method 1660
Load( ) method (Class) 2183
LoadControls method 1651
LOB-DIR option
INPUT FROM statement 727
OUTPUT TO statement 880
LOCAL-HOST attribute 1661
LOCAL-NAME attribute 1662
LOCAL-PORT attribute 1662
LOCAL-VERSION-INFO attribute 1662
LOCATOR-COLUMN-NUMBER attribute 1663
LOCATOR-LINE-NUMBER attribute 1663
LOCATOR-PUBLIC-ID attribute 1663
LOCATOR-SYSTEM-ID attribute 1664
LOCATOR-TYPE attribute 1664
LOCK-REGISTRATION method 1664
LOCKED attribute 1664
LOCKED function 789
LOG function 790
LOG-AUDIT-EVENT method 1665
LOG-ENTRY-TYPES attribute 1666
LOG-MANAGER system handle
attributes 1309
methods 1309
LOG-THRESHOLD attribute 1669
LOGFILE-NAME attribute 1670
LOGGING-LEVEL attribute 1671
LOGICAL data type 318
LOGICAL function 791
Logical values 792
LOGIN-EXPIRATION-TIMESTAMP attribute 1672
LOGIN-HOST attribute 1672
LOGIN-STATE attribute 1673
LoginDenied property 2184
LoginLockout property 2184
LOGOUT method 1675
LONG parameter data type 429
LONGCHAR data type 318
LONGCHAR-TO-NODE-VALUE method 1675
LOOKAHEAD option
QUERY-TUNING phrase 967
LOOKUP function 793
LOOKUP method 1676
LT operator 794

M

MANDATORY attribute 1676
MANUAL-HIGHLIGHT attribute 1676
MAP option
FUNCTION statement 671
INPUT FROM statement 727
INPUT THROUGH statement 733
INPUT-OUTPUT THROUGH statement 738
OUTPUT THROUGH statement 876
OUTPUT TO statement 880
MapNullToEmptyString property 2185
MARK-NEW method 1677
MARK-ROW-STATE method 1677
MATCHES operator 795
MAX-BUTTON attribute 1678
MAX-CHARS attribute 1679
MAX-CHARS option
COMBO-BOX phrase 215
EDITOR phrase 564
MAX-DATA-GUESS attribute 1679
MAX-HEIGHT-CHARS attribute 1680
MAX-HEIGHT-PIXELS attribute 1680
MAX-ROWS option
OPEN QUERY statement 857
MAX-SIZE option
SYSTEM-DIALOG FONT statement 1107
MAX-VALUE attribute 1680
MAX-VALUE option
SLIDER phrase 1085
MAX-WIDTH-CHARS attribute 1681
MAX-WIDTH-PIXELS attribute 1681
MaxDataGuess property 2185
MAXIMUM function 796
MAXIMUM option
aggregate phrase 111
MAXIMUM-LEVEL attribute 1680
MaxTries property 2186
MD5-DIGEST function 797
MD5-VALUE attribute 1681
MEMBER function 798
Members of an object type 315
MEMPTR data type 319
MEMPTR-TO-NODE-VALUE method 1681
Menu element descriptor
DEFINE MENU statement 423
DEFINE SUB-MENU statement 471
Menu item phrase
DEFINE MENU statement 423
DEFINE SUB-MENU statement 471
MENU option
DEFINE MENU statement 423
Widget phrase 1201

MENU widget
attributes 1239
dynamic 299
events 1239
methods 1239
MENU-BAR attribute 1682
MENU-DROP event
handling 848
MENU-ITEM option
DEFINE MENU statement 423
DEFINE SUB-MENU statement 471
Widget phrase 1201
MENU-ITEM widget
attributes 1240
dynamic 299
events 1240
methods 1240
MENU-KEY attribute 1682
MENU-MOUSE attribute 1682
MENUBAR option
DEFINE MENU statement 423
Menus
using CHOOSE 168
MERGE fill mode 1540
MERGE-BY-FIELD attribute 1683
MERGE-CHANGES method 1683
MERGE-ROW-CHANGES method 1684
MESSAGE option
HIDE statement 710
PAUSE statement 912
MESSAGE statement 799
MESSAGE-AREA attribute 1685
MESSAGE-AREA-FONT attribute 1685
MESSAGE-DIGEST function 806
MESSAGE-LINES function 807
MESSAGES option
COLOR phrase 210
Method body
METHOD statement 808
Method call
class-based 177
METHOD statement 808
Methods
class-based
parameter and return type descriptions 2052
handle-based object
invoking 1357
Methods and attributes
handle-based object 1353
Methods in classes
parameter and return type descriptions 2052
Progress.Data.BindingSource class 1968
Progress.Data.CancelCreateRowEventArgs class 1969
Progress.Data.ColumnPropDesc class 1971
Progress.Data.CreateRowEventArgs class 1972
Methods in classes *(continued)*
Progress.Data.DummyRow class 1974
Progress.Data.InputValue class 1975
Progress.Data.OffEndEventArgs class 1976
Progress.Data.SortRequestEventArgs class 1977
Progress.Data.TableDesc class 1978
Progress.Database.TempTableInfo class 1979
Progress.Database.VSTTableId class 1980
Progress.IO.BinarySerializer class 1981
Progress.IO.FileInputStream class 1982
Progress.IO.FileOutputStream class 1983
Progress.IO.InputStream class 1984
Progress.IO.JsonSerializer class 1985
Progress.IO.MemoryInputStream class 1986
Progress.IO.MemoryOutputStream class 1989
Progress.IO.OutputStream class 1992
Progress.Json.JsonError class 1992
Progress.Json.JsonParser class 1993
Progress.Json.JsonParserError class 1994
Progress.Json.ObjectModel.JsonArray class 1994
Progress.Json.ObjectModel.JsonConstruct class 1996
Progress.Json.ObjectModel.JsonDataType class 1997
Progress.Json.ObjectModel.JsonObject class 1998
Progress.Json.ObjectModel.ObjectModelParser class 1999
Progress.Lang.AppError class 2000
Progress.Lang.Class class 2002
Progress.Lang.Enum class 2005
Progress.Lang.FlagsEnum class 2006
Progress.Lang.LockConflict class 2007
Progress.Lang.Object class 2008
Progress.Lang.OERequestInfo class 2009
Progress.Lang.OEVersionInfo class 2010
Progress.Lang.ParameterList class 2011
Progress.Lang.ProError class 2012
Progress.Lang.SoapFaultError class 2013
Progress.Lang.Stop class 2014
Progress.Lang.StopAfter class 2015
Progress.Lang.StopError class 2016
Progress.Lang.SysError class 2017
Progress.Lang.UserInterrupt class 2018
Progress.Reflect.Constructor class 2020
Progress.Reflect.Event class 2022
Progress.Reflect.Method class 2024
Progress.Reflect.Parameter class 2025
Progress.Reflect.Property class 2026
Progress.Reflect.Variable class 2026
Progress.Security.PAMStatus class 2027
Progress.Util.DateTimeHelper class 2028
Progress.Util.DecimalHelper class 2029
Progress.Util.EnumHelper class 2030
Progress.Util.ResourceHelper class 2031
Progress.Util.StringHelper class 2032
Progress.Util.TypeHelper class 2033

Methods in classes *(continued)*
Progress.Windows.Form class *2034*
Progress.Windows.FormProxy class *2038*
Progress.Windows.MDIChildForm class *2040*
Progress.Windows.UserControl class *2045*
Progress.Windows.WindowContainer class *2048*
System.Exception class *2049*
Methods in interfaces
Progress.Lang.Error interface *2005*
Progress.Web.IWebHandler interface *2034*
Progress.Windows.IForm interface *2039*
Methods on handles
AUDIT-CONTROL system handle *1261*
AUDIT-POLICY system handle *1262*
Buffer object handle *1263*
Call object handle *1266*
Client-principal object handle *1273*
COLOR-TABLE system handle *1285*
COMPILER system handle *1288*
Data-source object handle *1292*
DEBUGGER system handle *1293*
ERROR-STATUS system handle *1299*
FONT-TABLE system handle *1305*
procedure object handle *1309*
ProDataSet object handle *1311*
PROFILER system handle *1313*
Query object handle *1315*
SAX-attributes object handle *1318*
SAX-reader object handle *1319*
SAX-writer object handle *1320*
SECURITY-POLICY system handle *1321*
Server object handle *1325*
Server socket object handle *1326*
SESSION system handle *1327*
SOAP-fault-detail object handle *1333*
SOAP-header object handle *1334*
Socket object handle *1336*
SOURCE-PROCEDURE system handle *1337*
TARGET-PROCEDURE system handle *1340*
Temp-table object handle *1343*
THIS-PROCEDURE system handle *1345*
Transaction object handle *1348*
WEB-CONTEXT system handle *1349*
X-document object handle *1350*
X-noderef object handle *1351*
Methods on widgets
BROWSE widget *1208*
BUTTON widget *1218*
COMBO-BOX widget *1220*
CONTROL-FRAME widget *1222*
DIALOG-BOX widget *1225*
EDITOR widget *1227*
FIELD-GROUP widget *1230*
FILL-IN widget *1231*
FRAME widget *1233*
IMAGE widget *1237*
Methods on widgets *(continued)*
LITERAL widget *1238*
MENU widget *1239*
MENU-ITEM widget *1240*
RADIO-SET widget *1241*
RECTANGLE widget *1243*
SELECTION-LIST widget *1244*
SHADOW-WINDOW widget *1246*
SLIDER widget *1247*
SUB-MENU widget *1249*
TEXT widget *1250*
TOGGLE-BOX widget *1251*
WINDOW widget *1253*
MIDDLE-MOUSE-CLICK event *1946*
MIDDLE-MOUSE-DBLCLICK event *1946*
MIDDLE-MOUSE-DOWN event *1946*
MIDDLE-MOUSE-UP event *1946*
MIN-BUTTON attribute *1686*
MIN-COLUMN-WIDTH-CHARS attribute *1686*
MIN-COLUMN-WIDTH-PIXELS attribute *1687*
MIN-HEIGHT-CHARS attribute *1687*
MIN-HEIGHT-PIXELS attribute *1687*
MIN-SCHEMA-MARSHAL attribute *1687*
MIN-SIZE option
COMPILE statement *222*
SYSTEM-DIALOG FONT statement *1107*
MIN-VALUE attribute *1688*
MIN-VALUE option
SLIDER phrase *1085*
MIN-WIDTH-CHARS attribute *1688*
MIN-WIDTH-PIXELS attribute *1688*
MINIMUM function *816*
MINIMUM option
aggregate phrase *111*
MissingCredentials property *2186*
Mode property *2186*
MODIFIED attribute *1689*
MODULO operator *817*
MONTH function *818*
Mouse events
portable *1944*
three-button *1945*
MOUSE-EXTEND-CLICK event *1944*
MOUSE-EXTEND-DBLCLICK event *1944*
MOUSE-EXTEND-DOWN event *1944*
MOUSE-EXTEND-UP event *1944*
MOUSE-MENU-CLICK event *1944*
MOUSE-MENU-DBLCLICK event *1944*
MOUSE-MENU-DOWN event *1944*
MOUSE-MENU-UP event *1944*
MOUSE-MOVE-CLICK event *1945*
MOUSE-MOVE-DBLCLICK event *1945*
MOUSE-MOVE-DOWN event *1945*
MOUSE-MOVE-UP event *1945*
MOUSE-POINTER attribute *1689*

MOUSE-POINTER option
DEFINE BUTTON statement *386*
DEFINE VARIABLE statement *490*
MOUSE-SELECT-CLICK event *1944*
MOUSE-SELECT-DBLCLICK event *1944*
MOUSE-SELECT-DOWN event *1944*
MOUSE-SELECT-UP event *1944*
MOVABLE attribute *1689*
MOVE-AFTER-TAB-ITEM method *1690*
MOVE-BEFORE-TAB-ITEM method *1690*
MOVE-COLUMN method *1691*
MOVE-TO-BOTTOM method *1692*
MOVE-TO-EOF method *1692*
MOVE-TO-TOP method *1692*
MstrBlkId property *2187*
MTIME function *819*
MULTI-COMPILE attribute *1693*
Multi-line comments (/* */) *84*
MULTIPLE attribute *1693*
MULTIPLE option
DEFINE BROWSE statement *366*
SELECTION-LIST phrase *1061*
MULTIPLE-KEY option
SYSTEM-HELP statement *1115*
Multiplication assignment operator (*=) *101*
Multiplication operator (*) *100*
MULTITASKING-INTERVAL attribute *1694*
MUST-EXIST option
SYSTEM-DIALOG GET-FILE statement *1110*
MUST-UNDERSTAND attribute *1695*

N

NAME attribute *1695*
Name property *1695*
Name property (Reflect) *2187*
Named events *941*, *1096*, *1155*, *1734*
Namespace
package *1167*
NAMESPACE-PREFIX attribute *1696*
NAMESPACE-PREFIX option
DEFINE BUFFER statement *380*
DEFINE DATASET statement *393*
DEFINE TEMP-TABLE statement *477*
NAMESPACE-URI attribute *1697*
NAMESPACE-URI option
DEFINE BUFFER statement *380*
DEFINE DATASET statement *393*
DEFINE TEMP-TABLE statement *477*
NATIVE fill-ins *1860*
NATIVE option
VIEW-AS phrase *1180*
Navigation key function events *1941*
NE operator *820*
NEEDS-APPSERVER-PROMPT attribute *1697*
NEEDS-PROMPT attribute *1697*
NESTED attribute *1697*
NESTED option
DEFINE DATASET statement *393*
.NET
arrays of mapped types *315*
data types *315*
default match *317*
mapped data types *315*
mapped object types *315*
object types *315*
primitive types *315*
.NET events
defining handlers *2273*
using *2274*
NEW attribute *1698*
NEW function (classes) *821*
NEW function (record buffers) *825*
NEW GLOBAL SHARED option
DEFINE TEMP-TABLE statement *477*
NEW GLOBAL SHARED STREAM option
DEFINE STREAM statement *469*
NEW GLOBAL SHARED VARIABLE option
DEFINE VARIABLE statement *490*
NEW option
LOAD statement *785*
NEW SHARED FRAME option
DEFINE FRAME statement *410*
NEW SHARED option
DEFINE BROWSE statement *366*
DEFINE BUFFER statement *380*
DEFINE DATASET statement *393*
DEFINE MENU statement *423*
DEFINE TEMP-TABLE statement *477*
NEW SHARED QUERY option
DEFINE QUERY statement *460*
NEW SHARED STREAM option
DEFINE STREAM statement *469*
NEW SHARED VARIABLE option
DEFINE VARIABLE statement *490*
NEW SHARED WORK-TABLE option
DEFINE WORK-TABLE statement *504*
NEW SHARED WORKFILE option
DEFINE WORK-TABLE statement *504*
NEW statement *826*
NEW-INSTANCE option
CREATE DATABASE statement *285*
NEW-ROW attribute *1698*
New( ) method *2187*
NewRow property *2188*
NEXT option
FIND statement *609*
GET statement *682*
ON ENDKEY phrase *841*
ON ERROR phrase *843*
ON QUIT phrase *846*
ON STOP phrase *854*

NEXT option *(continued)*
UNDO statement *1149*
NEXT statement *828*
NEXT-COLUMN attribute *1698*
NEXT-FRAME event *1942*
NEXT-PROMPT statement *828*
NEXT-ROWID attribute *1698*
NEXT-SIBLING attribute *1699*
NEXT-SIBLING property *2189*
NEXT-TAB-ITEM attribute *1701*
NEXT-VALUE function *830*
NextForm property *2188*
NO-APPLY option
ON ENDKEY phrase *841*
ON ERROR phrase *843*
ON QUIT phrase *846*
ON STOP phrase *854*
RETURN statement *1012*
UNDO statement *1149*
NO-ARRAY-MESSAGE option
QUERY-TUNING phrase *967*
NO-ASSIGN option
DEFINE BROWSE statement *366*
NO-ATTR-LIST option
INPUT FROM statement *727*
NO-AUTO-VALIDATE option
DEFINE BROWSE statement *366*
frame phrase *647*
NO-BIND-WHERE option
QUERY-TUNING phrase *967*
NO-BOX option
DEFINE BROWSE statement *366*
EDITOR phrase *564*
frame phrase *647*
NO-CONSOLE option
OS-COMMAND statement *866*
NO-CONVERT option
COPY-LOB statement *264*
INPUT FROM statement *727*
INPUT THROUGH statement *733*
INPUT-OUTPUT THROUGH statement *738*
OUTPUT THROUGH statement *876*
OUTPUT To statement *880*
NO-CONVERT-3D-COLOR option
DEFINE BUTTON statement *386*
NO-CURRENT-VALUE attribute *1701*
NO-CURRENT-VALUE option
SLIDER phrase *1085*
NO-DEBUG option
QUERY-TUNING phrase *967*
NO-DRAG option
SELECTION-LIST phrase *1061*
NO-ECHO option
INPUT FROM statement *727*
INPUT THROUGH statement *733*
INPUT-OUTPUT THROUGH statement *738*
NO-ECHO option *(continued)*
OUTPUT THROUGH statement *876*
OUTPUT TO statement *880*
NO-EMPTY-SPACE attribute *1701*
NO-EMPTY-SPACE option
DEFINE BROWSE statement *366*
NO-ERROR option
ASSIGN statement *126*
Assignment (=) statement *122*
BUFFER-COMPARE statement *144*
BUFFER-COPY statement *146*
CHOOSE statement *168*
COMPILE statement *222*
CONNECT statement *253*
COPY-LOB statement *264*
CREATE ALIAS statement *271*
CREATE automation object statement *274*
CREATE CALL statement *283*
CREATE DATABASE statement *285*
CREATE SAX-ATTRIBUTES statement *290*
CREATE SAX-READER statement *293*
CREATE SAX-WRITER statement *294*
CREATE SERVER-SOCKET statement *295*
CREATE SOCKET statement *297*
CREATE statement *268*
CREATE WIDGET-POOL statement *302*
DDE ADVISE statement *352*
DDE EXECUTE statement *354*
DDE GET statement *356*
DDE INITIATE statement *358*
DDE REQUEST statement *359*
DDE SEND statement *361*
DDE TERMINATE statement *363*
DELETE OBJECT statement *513*
DELETE PROCEDURE statement *517*
DELETE statement *510*
DELETE WIDGET-POOL statement *521*
DISCONNECT statement *530*
DISPLAY statement *531*
DYNAMIC-NEW statement *555*
EMPTY TEMP-TABLE statement *567*
EXTENT statement *603*
FIND statement *609*
IMPORT statement *717*
INSERT statement *742*
LOAD statement *785*
NEW statement *826*
PUT-KEY-VALUE statement *958*
RAW-TRANSFER statement *980*
RELEASE OBJECT statement *999*
RELEASE statement *997*
RUN statement *1023*
RUN STORED-PROCEDURE statement *1041*
RUN SUPER statement *1044*
SAVE CACHE statement *1047*
SET statement *1064*

NO-ERROR option *(continued)*
SUBSCRIBE statement *1096*
UNLOAD statement *1154*
UPDATE statement *1157*
USE statement *1164*
VALIDATE statement *1173*
NO-FILL fill mode *1540*
NO-FILL option
DEFINE RECTANGLE statement *465*
NO-FOCUS attribute *1702*
NO-FOCUS option
DEFINE BUTTON statement *386*
NO-HELP option
frame phrase *647*
NO-HIDE option
frame phrase *647*
NO-INDEX-HINT option
QUERY-TUNING phrase *967*
NO-INHERIT-BGCOLOR option
frame phrase *647*
NO-INHERIT-FGCOLOR option
frame phrase *647*
NO-JOIN-BY-SQLDB option
QUERY-TUNING phrase *967*
NO-LABELS option
DEFINE BROWSE statement *366*
format phrase *638*
frame phrase *647*
NO-LOBS option
BUFFER-COMPARE statement *144*
BUFFER-COPY statement *146*
EXPORT statement *594*
IMPORT statement *717*
NO-LOCK option
CAN-FIND function *155*
DEFINE BROWSE statement *366*
FIND statement *609*
FIND-BY-ROWID method *1541*
GET statement *682*
GET-CURRENT method *1570*
GET-FIRST method *1574*
GET-LAST method *1577*
GET-NEXT method *1580*
GET-PREV method *1582*
Record phrase *985*
NO-LOOKAHEAD option
QUERY-TUNING phrase *967*
NO-MAP option
INPUT FROM statement *727*
INPUT THROUGH statement *733*
INPUT-OUTPUT THROUGH statement *738*
OUTPUT THROUGH statement *876*
OUTPUT TO statement *880*
NO-MESSAGE option
PAUSE statement *912*
NO-PAUSE option
CLEAR statement *204*
HIDE statement *710*
NO-PREFETCH option
CAN-FIND function *155*
FIND statement *609*
Record phrase *985*
NO-ROW-MARKERS option
DEFINE BROWSE statement *366*
NO-SCHEMA-MARSHAL attribute *1703*
NO-SCROLLBAR-VERTICAL option
DEFINE BROWSE statement *366*
NO-SEPARATE-CONNECTION option
QUERY-TUNING phrase *967*
NO-SEPARATORS option
DEFINE BROWSE statement *366*
NO-TAB-STOP option
format phrase *638*
NO-UNDERLINE option
frame phrase *647*
NO-UNDO option
DEFINE PARAMETER statement *429*
DEFINE PROPERTY statement *442*
DEFINE TEMP-TABLE statement *477*
DEFINE VARIABLE statement *490*
DEFINE WORK-TABLE statement *504*
NO-VALIDATE attribute *1703*
NO-VALIDATE option
DEFINE BROWSE statement *366*
frame phrase *647*
NO-WAIT option
CAN-FIND function *155*
DEFINE BROWSE statement *366*
FIND statement *609*
FIND-BY-ROWID method *1541*
GET statement *682*
GET-CURRENT method *1570*
GET-FIRST method *1574*
GET-LAST method *1577*
GET-NEXT method *1580*
GET-PREV method *1582*
OS-COMMAND statement *866*
NO-WORD-WRAP option
EDITOR phrase *564*
NODE-VALUE attribute *1704*
NODE-VALUE-TO-LONGCHAR method *1704*
NODE-VALUE-TO-MEMPTR method *1705*
NON-SERIALIZABLE option
DEFINE PROPERTY statement *442*
DEFINE TEMP-TABLE statement *477*
DEFINE VARIABLE statement *490*
VAR statement *1174*
NON-SERIALIZABLE optionDATASET option
DEFINE DATASET statement *393*
NONAMESPACE-SCHEMA-LOCATION attribute *1703*
NORMAL menu items *1860*

NORMAL option
COLOR phrase *210*
NORMALIZE function *832*
NORMALIZE method *1706*
NOT CASE-SENSITIVE option
DEFINE VARIABLE statement *490*
NOT ENTERED function *834*
Not Equal to operator *820*
NOT operator (bitwise) *832*
NOT operator (logical) *833*
NOT-ACTIVE option
DEFINE DATASET statement *393*
NOW function *835*
Null property *2190*
NUM-ALIASES function *836*
NUM-BUFFERS attribute *1706*
NUM-BUTTONS attribute *1706*
NUM-CHILD-RELATIONS attribute *1706*
NUM-CHILDREN attribute *1707*
NUM-COLUMNS attribute *1707*
NUM-COPIES option
OUTPUT TO statement *880*
SYSTEM-DIALOG PRINTER-SETUP statement *1114*
NUM-DBS function *837*
NUM-DROPPED-FILES attribute *1707*
NUM-ENTRIES attribute *1707*
NUM-ENTRIES function *837*
NUM-FIELDS attribute *1708*
NUM-FORMATS attribute *1708*
NUM-HEADER-ENTRIES attribute *1708*
NUM-ITEMS attribute *1708*
NUM-ITERATIONS attribute *1709*
NUM-LINES attribute *1709*
NUM-LOCKED-COLUMNS attribute *1709*
NUM-LOG-FILES attribute *1710*
NUM-MESSAGES attribute *1710*
NUM-PARAMETERS attribute *1711*
NUM-REFERENCES attribute *1711*
NUM-RELATIONS attribute *1712*
NUM-REPLACED attribute *1712*
NUM-RESULTS attribute *1712*
NUM-RESULTS function *839*
NUM-SELECTED-ROWS attribute *1712*
NUM-SELECTED-WIDGETS attribute *1713*
NUM-SOURCE-BUFFERS attribute *1713*
NUM-TABS attribute *1713*
NUM-TO-RETAIN attribute *1713*
NUM-TOP-BUFFERS attribute *1713*
NUM-VISIBLE-COLUMNS attribute *1714*
Number property *2190*
NUMERIC-DECIMAL-POINT attribute *1714*
NUMERIC-FORMAT attribute *1714*
NUMERIC-SEPARATOR attribute *1714*
NumHandlerParameters property *2190*
NumMessages property *2190*
NumParameters property (ParameterList) *2191*
NumParameters property (Reflect) *2191*

O

OBJECT option (source)
COPY-LOB statement *264*
OBJECT option (target)
COPY-LOB statement *264*
Object property *2191*
Object references
defined *315*
using *180*
validating *1173*
Object types
ABL *315*
.NET *315*
Objects
class-based
built-in *1961*
user-defined *185*
handle-based *1257*
OEClientType property *2191*
OEMaintVersion property *2192*
OEMajorVersion property *2192*
OEMinorVersion property *2192*
OF option
CAN-FIND function *155*
FIND statement *609*
Record phrase *985*
OFF option
PUT CURSOR statement *944*
OFF-END event
GUI *1949*
ProDataSet *1958*
OFF-HOME event *1949*
OffEnd event (.NET) *2278*
Offset property *2193*
ON ENDKEY phrase
DO statement *537*
error-object-expression *841*
FOR statement *620*
REPEAT statement *1001*
ON ERROR phrase
DO statement *537*
error-object-expression *843*
FOR statement *620*
REPEAT statement *1001*
ON QUIT phrase
DO statement *537*
error-object-expression *846*
FOR statement *620*
REPEAT statement *1001*
ON SERVER option
RUN statement
local procedure call *1023*

ON SERVER option *(continued)*
RUN statement *(continued)*
remote procedure call *1023*
SESSION handle asynchronous *1023*
SESSION handle synchronous *1023*
ON statement *848*
ON STOP phrase
DO statement *537*
error-object-expression *854*
FOR statement *620*
REPEAT statement *1001*
ON-FRAME-BORDER attribute *1715*
Open generic types *317*
OPEN QUERY statement *857*
Operating system devices
INPUT FROM statement *727*
Operating system files
INPUT FROM statement *727*
OPSYS function *862*
OPSYS preprocessor name *74*
OPTIONS attribute *1715*
Options omitted
CREATE automation object statement *275*
OPTIONS option
COMPILE statement *222*
OPTIONS-FILE option
COMPILE statement *222*
OR operator (bitwise) *863*
OR operator (logical) *863*
Or( ) method *2193*
ORDERED-JOIN option
QUERY-TUNING phrase *967*
ORDINAL attribute *1715*
ORDINAL option
PROCEDURE statement *919*
ORIGIN-HANDLE attribute *1716*
ORIGIN-ROWID attribute *1716*
OriginatingClass property *2194*
OS-APPEND statement *864*
OS-COMMAND statement *866*
OS-COPY statement *867*
OS-CREATE-DIR statement *868*
OS-DELETE statement *869*
OS-DIR option
INPUT FROM statement *727*
OS-DRIVES function *871*
OS-ERROR function *871*
OS-GETENV function *873*
OS-RENAME statement *874*
_osprint.p file *880*
OTHERWISE option
CASE statement *161*
OUT preprocessor name *74*
OUT-FMT preprocessor name *74*
OUT-LONG preprocessor name *74*
OUTER-JOIN option
Record phrase *985*
OUTPUT CLOSE statement *875*
OUTPUT option
Parameter definition syntax *892*
Parameter passing syntax *898*
SEEK function *1058*
SEEK statement *1059*
stored procedures *1041*
OUTPUT PARAMETER option
DEFINE PARAMETER statement *429*
OUTPUT THROUGH statement *876*
OUTPUT TO statement *880*
OVERLAY AT option
COPY-LOB statement *264*
OVERLAY attribute *1716*
OVERLAY option
frame phrase *647*
OVERLAY statement *888*
OVERRIDE option
DEFINE EVENT statement *404*
DEFINE PROPERTY statement *442*
METHOD statement *808*
ON statement *848*
OWNER attribute *1717*
OWNER-DOCUMENT attribute *1717*

P

Package
namespace *1167*
type-name syntax *1140*
Package property *2195*
PACKAGE-PRIVATE option
CONSTRUCTOR statement *260*
DEFINE EVENT statement *404*
DEFINE PROPERTY statement *442*
DEFINE VARIABLE statement *490*
METHOD statement *808*
VAR statement *1174*
PACKAGE-PROTECTED option
CONSTRUCTOR statement *260*
DEFINE EVENT statement *404*
DEFINE PROPERTY statement *442*
DEFINE VARIABLE statement *490*
METHOD statement *808*
VAR statement *1174*
PAGE statement *890*
PAGE-BOTTOM attribute *1717*
PAGE-BOTTOM option
frame phrase *647*
PAGE-NUMBER function *891*
PAGE-SIZE function *891*
PAGE-SIZE option
COMPILE statement *222*
OUTPUT THROUGH statement *876*

PAGE-SIZE option *(continued)*
OUTPUT TO statement *880*
PAGE-TOP attribute *1717*
PAGE-TOP option
frame phrase *647*
rules *647*
PAGE-WIDTH option
COMPILE statement *222*
PAGED option
OUTPUT THROUGH statement *876*
OUTPUT TO statement *880*
PARAM option
stored procedures *1041*
PARAMETER attribute *1718*
PARAMETER BUFFER option
DEFINE PARAMETER statement *429*
Parameter definition
CONSTRUCTOR statement *260*
DEFINE EVENT statement *404*
FUNCTION statement *671*
METHOD statement *808*
SUPER function *1103*
Parameter definition syntax *892*
Parameter passing
default mode *899*
NEW function *555*, *821*
PUBLISH statement *941*
RUN statement *1023*
RUN SUPER statement *1044*
SUPER statement *1104*
SUPER system reference *1105*
THIS-OBJECT statement *1124*
Parameter passing syntax *898*
PARENT attribute *1718*
PARENT-BUFFER attribute *1719*
PARENT-FIELDS-AFTER attribute *1719*
PARENT-FIELDS-AFTER option
DEFINE DATASET statement *393*
PARENT-FIELDS-BEFORE attribute *1719*
PARENT-FIELDS-BEFORE option
DEFINE DATASET statement *393*
PARENT-ID-FIELD option
DEFINE DATASET statement *393*
PARENT-ID-RELATION attribute *1720*
PARENT-ID-RELATION option
DEFINE DATASET statement *393*
PARENT-RELATION attribute *1719–1720*
PARENT-WINDOW-CLOSE Event *1949*
Parentheses (( ))
expression precedence *66*
PARSE-STATUS attribute *1720*
Parse( ) method *2196*
ParseFile( ) method *2196*
PARTIAL-KEY option
SYSTEM-HELP statement *1115*
PASCAL option
PROCEDURE statement *919*
Passing parameters *898*
PASSWORD-FIELD attribute *1721*
PasswordExpired property *2197*
PATHNAME attribute *1721*
PAUSE option
CHOOSE statement *168*
READKEY statement *982*
WAIT-FOR statement *1186*
PAUSE statement *912*
PBE-HASH-ALGORITHM attribute *1722*
PBE-KEY-ROUNDS attribute *1722*
PDBNAME function *914*
Performance
FORWARD-ONLY attribute *1557*
Period (.)
punctuation *62*
PermissionDenied property *2197*
PERSISTENT attribute *1722*
PERSISTENT option
CREATE WIDGET-POOL statement *302*
PROCEDURE statement *919*
RUN statement *1023*
PERSISTENT RUN option
ON statement *848*
Trigger phrase *1131*
PERSISTENT-CACHE-DISABLED attribute *1723*
PERSISTENT-PROCEDURE attribute *1723*
PFCOLOR attribute *1723*
PFCOLOR option
DEFINE BROWSE statement *366*
DEFINE BUTTON statement *386*
DEFINE FRAME statement *410*
DEFINE MENU statement *423*
DEFINE RECTANGLE statement *465*
DEFINE SUB-MENU statement *471*
DEFINE VARIABLE statement *490*
ENABLE statement *568*
FORM statement *632*
format phrase *638*
frame phrase *647*
PIXELS-PER-COLUMN attribute *1724*
PIXELS-PER-ROW attribute *1724*
POPUP-MENU attribute *1724*
POPUP-ONLY attribute *1725*
PORTRAIT option
OUTPUT TO statement *880*
SYSTEM-DIALOG PRINTER-SETUP statement *1114*
POSITION attribute *1725*
POSITION MAXIMIZE option
SYSTEM-HELP statement *1115*
Position property (BindingSource) *2197*
Position property (MemoryInputStream) *2198*
Position property (Parameter) *2198*

POSITION X Y WIDTH HEIGHT option
SYSTEM-HELP statement 1115
PositionChanged event 2279
PREFER-DATASET attribute 1725
PREPARE-STRING attribute 1726
PREPARED attribute 1726
PREPROCESS option
COMPILE statement 222
Preprocessor
argument reference 68
built-in names 74
Preprocessor name reference 74
PRESELECT option
DEFINE BUFFER statement 380
DEFINE PARAMETER statement 429
OPEN QUERY statement 857
Parameter definition syntax 892
PRESELECT phrase
DO statement 537
REPEAT statement 1001
PREV keyword value
DBRESTRICTIONS function 347
PREV option
FIND statement 609
GET statement 682
PREV-COLUMN attribute 1726
PREV-FRAME event 1943
PREV-SIBLING attribute 1726
PREV-SIBLING property 2199
PREV-TAB-ITEM attribute 1727
PrevForm property 2198
PRIMARY attribute 1728
PRIMARY option
DEFINE TEMP-TABLE statement 477
PRIMARY-PASSPHRASE attribute 1728
Primitive types
ABL 315
.NET 315
PRINTER option
OUTPUT TO statement 880
PRINTER-CONTROL-HANDLE attribute 1729
PRINTER-HDC attribute 1729
PRINTER-NAME attribute 1730
PRINTER-PORT attribute 1730
Printing
_osprint.p file 880
OUTPUT THROUGH statement 876
OUTPUT TO statement 880
PRINTER-CONTROL- HANDLE attribute 1729
SYSTEM-DIALOG PRINTER-SETUP statement 1114
PRIVATE option
CONSTRUCTOR statement 260
DEFINE BROWSE statement 366
DEFINE BUFFER statement 380
DEFINE BUTTON statement 386
PRIVATE option *(continued)*
DEFINE DATA-SOURCE statement 401
DEFINE DATASET statement 393
DEFINE EVENT statement 404
DEFINE FRAME statement 410
DEFINE IMAGE statement 419
DEFINE MENU statement 423
DEFINE PROPERTY statement 442
DEFINE QUERY statement 460
DEFINE RECTANGLE statement 465
DEFINE STREAM statement 469
DEFINE SUB-MENU statement 471
DEFINE TEMP-TABLE statement 477
DEFINE VARIABLE statement 490
DEFINE WORK-TABLE statement 504
DEFINE WORKFILE statement 504
FUNCTION statement 671
METHOD statement 808
PROCEDURE statement 919
VAR statement 1174
PRIVATE-DATA attribute 1730
PROC-HANDLE function
CLOSE STORE-PROCEDURE statement 207
RUN STORED-PROCEDURE statement and 1041
PROC-STATUS function
CLOSE STORE-PROCEDURE statement 207
proc-text field name
DISPLAY statement 531
proc-text-buffer 917–918, 1041
proc-text-buffer name
DEFINE BUFFER statement 380
FOR statement 620
Procedure body
PROCEDURE statement 919
Procedure handle events
WAIT-FOR statement 1186
Procedure handles
RUN statement 1023
Procedure object handle
attributes 1309
methods 1309
PROCEDURE option
RELEASE EXTERNAL statement 999
SUBSCRIBE statement 1096
UNSUBSCRIBE statement 1155
Procedure overloading 1103, 1105, 1766, 1861
PROCEDURE statement 919
PROCEDURE-COMPLETE event
handling
multiple requests 1186
Progress actions 1186
PROCEDURE-NAME attribute 1731
PROCEDURE-TYPE attribute 1731
ProcedureName property 2199
Procedures
multi-line comments (/* */) 84

Procedures *(continued)*
single-line comments (//) *85*
PROCESS EVENTS statement *925*
PROCESS-ARCHITECTURE function *924*
ProDataSet events
FIND-FAILED *1959*
first-level FILL *1956*
OFF-END *1958*
row-level *1957*
second-level FILL *1957*
SYNCHRONIZE *1959*
ProDataSet object handle
attributes *1311*
events *1311*
methods *1311*
PROFILER system handle
attributes *1313*
methods *1313*
PROFILING attribute *1732*
PROGRAM-NAME function *926*
Progress fill-ins *1860*
PROGRESS function *928*
PROGRESS-SOURCE attribute *1732*
Progress.ApplicationServer.AdapterType enumeration *1963*
Progress.Data.BindingSource class
events *1968*
methods *1968*
properties *1968*
Progress.Data.CancelCreateRowEventArgs class
events *1969*
methods *1969*
properties *1969*
Progress.Data.ColumnPropDesc class
events *1971*
methods *1971*
properties *1971*
Progress.Data.CreateRowEventArgs class
events *1972*
methods *1972*
properties *1972*
Progress.Data.DataType enumeration class *1973*
Progress.Data.DummyRow class
events *1974*
methods *1974*
properties *1974*
Progress.Data.InputValue class
events *1975*
methods *1975*
properties *1975*
Progress.Data.OffEndEventArgs class
events *1976*
methods *1976*
properties *1976*
Progress.Data.SortRequestEventArgs class
events *1977*
Progress.Data.SortRequestEventArgs class *(continued)*
methods *1977*
properties *1977*
Progress.Data.TableDesc class
events *1978*
methods *1978*
properties *1978*
Progress.Database.TempTableInfo class
events *1979*
methods *1979*
properties *1979*
Progress.Database.VSTTableId class
events *1980*
methods *1980*
properties *1980*
Progress.IO.BinarySerializer class
events *1981*
methods *1981*
properties *1981*
Progress.IO.ByteOrder enumeration class *1982*
Progress.IO.FileInputStream class
events *1982*, *1986*
methods *1982*, *1986*
properties *1982*
Progress.IO.FileOutputStream class
events *1983*
methods *1983*
properties *1983*
Progress.IO.InputStream class
events *1984*
methods *1984*
properties *1984*
Progress.IO.JsonSerializer class
events *1985*
methods *1985*
properties *1985*
Progress.IO.MemoryInputStream class
properties *1986*
Progress.IO.MemoryOutputStream class
events *1989*
methods *1989*
properties *1989*
Progress.IO.OutputStream class
events *1992*
methods *1992*
properties *1992*
Progress.Json.JsonError class
events *1992*
methods *1992*
properties *1992*
Progress.Json.JsonParser class
events *1993*
methods *1993*
properties *1993*
Progress.Json.JsonParserError class
events *1994*

Progress.Json.JsonParserError class *(continued)*
methods *1994*
properties *1994*
Progress.Json.ObjectModel.JsonArray class
events *1994*
methods *1994*
properties *1994*
Progress.Json.ObjectModel.JsonConstruct class
events *1996*
methods *1996*
properties *1996*
Progress.Json.ObjectModel.JsonDataType class
events *1997*
methods *1997*
properties *1997*
Progress.Json.ObjectModel.JsonObject class
events *1998*
methods *1998*
properties *1998*
Progress.Json.ObjectModel.ObjectModelParser class
events *1999*
methods *1999*
properties *1999*
Progress.Lang.AppError class
events *2000*
methods *2000*
properties *2000*
Progress.Lang.Class class
events *2002*
methods *2002*
properties *2002*
Progress.Lang.Enum class
events *2005*
methods *2005*
properties *2005*
Progress.Lang.Error interface
methods *2005*
properties *2005*
Progress.Lang.FlagsEnum class
methods *2006*
Progress.Lang.LockConflict class
events *2007*
methods *2007*
properties *2007*
Progress.Lang.Object class
events *2008*
methods *2008*
properties *2008*
Progress.Lang.OERequestInfo class
events *2009*
methods *2009*
properties *2009*
Progress.Lang.OEVersionInfo class
events *2010*
methods *2010*
properties *2010*
Progress.Lang.ParameterList class
events *2011*
methods *2011*
properties *2011*
Progress.Lang.ProError class
events *2012*
methods *2012*
properties *2012*
Progress.Lang.SoapFaultError class
events *2013*
methods *2013*
properties *2013*
Progress.Lang.Stop class
events *2014*
methods *2014*
properties *2014*
Progress.Lang.StopAfter class
events *2015*
methods *2015*
properties *2015*
Progress.Lang.StopError class
events *2016*
methods *2016*
properties *2016*
Progress.Lang.SysError class
events *2017*
methods *2017*
properties *2017*
Progress.Lang.UserInterrupt class
events *2018*
methods *2018*
properties *2018*
Progress.Reflect.AccessMode enumeration *2019*
Progress.Reflect.Constructor class
events *2020*
methods *2020*
properties *2020*
Progress.Reflect.DataType enumeration *2021*
Progress.Reflect.Event class
events *2022*
methods *2022*
properties *2022*
Progress.Reflect.Flags enumeration *2022*
Progress.Reflect.Method class
events *2024*
methods *2024*
properties *2024*
Progress.Reflect.Parameter class
events *2025*
methods *2025*
properties *2025*
Progress.Reflect.ParameterMode enumeration *2025*
Progress.Reflect.Property class
events *2026*
methods *2026*
properties *2026*

Progress.Reflect.Variable class
events 2026
methods 2026
properties 2026
Progress.Security.PAMStatus class
events 2027
methods 2027
properties 2027
Progress.Util.DateTimeHelper class
events 2028
methods 2028
properties 2028
Progress.Util.DecimalHelper class
events 2029
methods 2029
properties 2029
Progress.Util.EnumHelper class
events 2030
methods 2030
properties 2030
Progress.Util.ResourceHelper class
events 2031
methods 2031
properties 2031
Progress.Util.StringHelper class
events 2032
methods 2032
properties 2032
Progress.Util.TypeHelper class
events 2033
methods 2033
properties 2033
Progress.Web.IWebHandler interface
methods 2034
properties 2034
Progress.Windows.Form class
events 2034
methods 2034
properties 2034
Progress.Windows.FormProxy class
events 2038
methods 2038
properties 2038
Progress.Windows.IForm interface
methods 2039
properties 2039
Progress.Windows.MDIChildForm class
events 2040
methods 2040
properties 2040
Progress.Windows.MDIChildRadForm class 2041
Progress.Windows.MDIChildRadRibbonForm class 2042
Progress.Windows.MDIChildShapedForm class 2043
Progress.Windows.OERadForm class 2043
Progress.Windows.OERadRibbonForm class 2044
Progress.Windows.OEShapedForm class 2044
Progress.Windows.UserControl class
events 2045
methods 2045
properties 2045
Progress.Windows.WindowContainer class
events 2048
methods 2048
properties 2048
PROMPT option
COLOR statement 213
frame phrase 647
PROMPT-FOR statement 929
PROMSGS function 936
PROMSGS statement 936
PROPATH function 937
PROPATH statement 938
Properties
name 442
Properties in classes
data type descriptions 2052
Progress.Data.BindingSource class 1968
Progress.Data.CancelCreateRowEventArgs class 1969
Progress.Data.ColumnPropDesc class 1971
Progress.Data.CreateRowEventArgs class 1972
Progress.Data.DummyRow class 1974
Progress.Data.InputValue class 1975
Progress.Data.OffEndEventArgs class 1976
Progress.Data.SortRequestEventArgs class 1977
Progress.Data.TableDesc class 1978
Progress.Database.TempTableInfo class 1979
Progress.Database.VSTTableId class 1980
Progress.IO.BinarySerializer class 1981
Progress.IO.FileInputStream class 1982
Progress.IO.FileOutputStream class 1983
Progress.IO.InputStream class 1984
Progress.IO.JsonSerializer class 1985
Progress.IO.MemoryInputStream class 1986
Progress.IO.MemoryOutputStream class 1989
Progress.IO.OutputStream class 1992
Progress.Json.JsonError class 1992
Progress.Json.JsonParser class 1993
Progress.Json.JsonParserError class 1994
Progress.Json.ObjectModel.JsonArray class 1994
Progress.Json.ObjectModel.JsonConstruct class 1996
Progress.Json.ObjectModel.JsonDataType class 1997
Progress.Json.ObjectModel.JsonObject class 1998
Progress.Json.ObjectModel.ObjectModelParser class 1999
Progress.Lang.AppError class 2000
Progress.Lang.Class class 2002
Progress.Lang.Enum class 2005
Progress.Lang.LockConflict class 2007
Progress.Lang.Object class 2008

Properties in classes *(continued)*
Progress.Lang.OERequestInfo class *2009*
Progress.Lang.OEVersionInfo class *2010*
Progress.Lang.ParameterList class *2011*
Progress.Lang.ProError class *2012*
Progress.Lang.SoapFaultError class *2013*
Progress.Lang.Stop class *2014*
Progress.Lang.StopAfter class *2015*
Progress.Lang.StopError class *2016*
Progress.Lang.SysError class *2017*
Progress.Lang.UserInterrupt class *2018*
Progress.Reflect.Constructor class *2020*
Progress.Reflect.Event class *2022*
Progress.Reflect.Method class *2024*
Progress.Reflect.Parameter class *2025*
Progress.Reflect.Property class *2026*
Progress.Reflect.Variable class *2026*
Progress.Security.PAMStatus class *2027*
Progress.Util.DateTimeHelper class *2028*
Progress.Util.DecimalHelper class *2029*
Progress.Util.EnumHelper class *2030*
Progress.Util.ResourceHelper class *2031*
Progress.Util.StringHelper class *2032*
Progress.Util.TypeHelper class *2033*
Progress.Windows.Form class *2034*
Progress.Windows.FormProxy class *2038*
Progress.Windows.MDIChildForm class *2040*
Progress.Windows.UserControl class *2045*
Progress.Windows.WindowContainer class *2048*
System.Exception class *2049*
Properties in interfaces
data type descriptions *2052*
Progress.Lang.Error interface *2005*
Progress.Web.IWebHandler interface *2034*
Progress.Windows.IForm interface *2039*
Property
accessing *181*
Property name
DEFINE PROPERTY statement *442*
PROTECTED option
CONSTRUCTOR statement *260*
DEFINE BUFFER statement *380*
DEFINE DATA-SOURCE statement *401*
DEFINE DATASET statement *393*
DEFINE EVENT statement *404*
DEFINE PROPERTY statement *442*
DEFINE QUERY statement *460*
DEFINE TEMP-TABLE statement *477*
DEFINE VARIABLE statement *490*
METHOD statement *808*
VAR statement *1174*
protermcap attributes *210*
PROVERSION function *940*
ProWinHandle property *2200*
PROXY attribute *1732*
PROXY-PASSWORD attribute *1733*
PROXY-USERID attribute *1733*
PUBLIC option
CONSTRUCTOR statement *260*
DEFINE EVENT statement *404*
DEFINE PROPERTY statement *442*
DEFINE VARIABLE statement *490*
DESTRUCTOR statement *523*
METHOD statement *808*
VAR statement *1174*
PUBLIC-ID attribute *1733*
PUBLISH statement *941*
Publish( ) method *2280*
PUBLISHED-EVENTS attribute *1734*
PUT CURSOR statement *944*
PUT SCREEN statement *946*
PUT statement *948*
PUT-BITS statement *952*
PUT-BYTE statement *953*
PUT-BYTES statement *955*
PUT-DOUBLE statement *955*
PUT-FLOAT statement *956*
PUT-INT64 statement *957*
PUT-KEY-VALUE statement *958*
PUT-LONG statement *962*
PUT-SHORT statement *963*
PUT-STRING statement *964*
PUT-UNSIGNED-LONG statement *965*
PUT-UNSIGNED-SHORT statement *966*

Q

QUALIFIED-USER-ID attribute *1734*
Queries
opening *857*
results lists *839*
QUERY attribute *1735*
Query object
ADD-BUFFER method *1364*
CACHE attribute *1419*
CREATE QUERY statement *289*
CURRENT-RESULT-ROW attribute *1481*
GET-LAST method *1577*
GET-NEXT method *1580*
GET-PREV method *1582*
INDEX-INFORMATION attribute *1614*
NUM-BUFFERS attribute *1706*
QUERY-CLOSE method *1736*
QUERY-OFF-END attribute *1734*, *1737*
QUERY-OPEN method *1737*
QUERY-PREPARE method *1738*
REPOSITION-TO-ROWID method *1772*
Query object handle
attributes *1315*
methods *1315*
QUERY option
DEFINE BROWSE statement *366*

QUERY option *(continued)*
DEFINE DATA-SOURCE statement *401*
DEFINE QUERY statement *460*
QUERY-CLOSE method *1736*
QUERY-OFF-END attribute *1737*
QUERY-OFF-END function *966*
QUERY-OPEN method *1737*
QUERY-PREPARE method *1738*
Query-tuning phrase
REPEAT statement *1001*
QUERY-TUNING phrase
DO statement *537*
FOR statement *620*
OPEN QUERY statement *857*
Question mark (?)
special character *63*
QUIT attribute *1739*
QUIT option
SYSTEM-HELP statement *1115*
QUIT statement
compared to ON QUIT phrase *846*
QUOTER function *971*

R

R option
Character-string literal (" ") *67*
R-code library members, running *1023*
R-INDEX function *973*
RADIO-BUTTONS attribute *1739*
RADIO-BUTTONS option
RADIO-SET phrase *975*
RADIO-SET phrase
VIEW-AS phrase *1180*
RADIO-SET widget
attributes *1241*
dynamic *299*
events *1241*
methods *1241*
RANDOM function *977*
RAW data type *319*
RAW function *978*
RAW statement *979*
RAW-TRANSFER method *1740*
RAW-TRANSFER statement *980*
RCODE-INFO system handle
attributes *1316*
RCODE-INFORMATION option
DEFINE QUERY statement *460*
DEFINE TEMP-TABLE statement *477*
READ method (Socket) *1740*
READ-FILE method *1741*
READ-JSON method *1742*
READ-ONLY attribute *1745*
READ-ONLY keyword value
DBRESTRICTIONS function *347*
READ-ONLY option
DEFINE MENU statement *423*
DEFINE SUB-MENU statement *471*
READ-RESPONSE event *1955*
READ-RESPONSE event *1955*
READ-XML method *1746*
READ-XMLSCHEMA method *1752*
Read( ) method (InputStream) *2201*
Read( ) method (JsonArray) *2202*
Read( ) method (JsonObject) *2203*
Read( ) method (MemoryInputStream) *2205*
ReadByte( ) method *2206*
ReadDouble( ) method *2207*
ReadFloat( ) method *2207*
ReadInt64( ) method *2208*
READKEY statement *982*
ReadLong( ) method *2208*
ReadLongchar( ) method *2208*
ReadOnly property *2210*
ReadShort( ) method *2209*
ReadString( ) method *2209*
ReadUnsignedLong( ) method *2209*
ReadUnsignedShort( ) method *2210*
RECALL event *1943*
RECID attribute *1757*
RECID data type *319*
RECID function *984*
RECID keyword value
DBRESTRICTIONS function *347*
Record phrase
FOR statement *620*
RECORD-LENGTH attribute *1757*
RECORD-LENGTH function *996*
RECTANGLE option
DEFINE RECTANGLE statement *465*
RECTANGLE widget
attributes *1243*
dynamic *299*
events *1243*
methods *1243*
RECURSIVE attribute *1757*
RECURSIVE option
DEFINE DATASET statement *393*
OS-DELETE statement *869*
REFERENCE-ONLY option
DEFINE DATASET statement *393*
DEFINE TEMP-TABLE statement *477*
REFRESH method (Handle) *1757*
REFRESH-AUDIT-POLICY method *1758*
Refresh( ) method (Progress.Data.BindingSource) *2210*
REFRESHABLE attribute *1758*
RefreshAll( ) method *2211*
REGISTER-DOMAIN method *1759*
REJECT-CHANGES method *1760*
REJECT-ROW-CHANGES method *1761*
REJECTED attribute *1761*

REJECTED function *997*
RELATION-FIELDS attribute *1761*
RELATION-FIELDS option
    DEFINE DATASET statement *393*
RELATIONS-ACTIVE attribute *1762*
RELEASE EXTERNAL statement *999*
RELEASE OBJECT statement *999*
RELEASE statement *997*
REMOTE attribute *1763*
Remote procedure handles, validating *1171*
Remote procedures
    asynchronous, obtaining results *919*
    calling *1023*
    obtaining return values *919*, *1015*
    RETURN statement *1012*
REMOTE-HOST attribute *1763*
REMOTE-PORT attribute *1763*
REMOVE-ATTRIBUTE method *1764*
REMOVE-CHILD method *1765*
REMOVE-EVENTS-PROCEDURE method *1765*
REMOVE-SUPER-PROCEDURE method *1766*
Remove( ) method (JsonArray) *2212*
Remove( ) method (JsonObject) *2213*
RemoveMessage( ) method *2213*
REPEAT statement *1001*
REPLACE fill mode *1540*
REPLACE function *1007*
REPLACE method *1767*
REPLACE option
    CREATE DATABASE statement *285*
REPLACE-CHILD method *1769*
REPLACE-SELECTION-TEXT method *1770*
REPOSITION attribute *1770*
REPOSITION option
    DEFINE DATASET statement *393*
REPOSITION statement *1008*
REPOSITION-BACKWARD method *1771*
REPOSITION-FORWARD method *1771*
REPOSITION-TO-ROW method *1772*
REPOSITION-TO-ROWID method *1772*
REQUEST-INFO attribute *1774*
RequestId property *2214*
Required: add at least one index entry *835*
RESET method *1775*
Reset( ) method (MemoryInputStream) *2214*
RESIZABLE attribute *1776*
RESIZE attribute *1776*
RESPONSE-INFO attribute *1776*
RESTART-ROW attribute *1778*
RESTART-ROWID attribute *1779*
Results lists *839*
RETAIN option
    frame phrase *647*
RETAIN-SHAPE attribute *1780*
RETAIN-SHAPE option
    DEFINE IMAGE statement *419*
RETRY function *1011*
RETRY option
    ON ENDKEY phrase *841*
    ON ERROR phrase *843*
    ON QUIT phrase *846*
    ON STOP phrase *854*
    UNDO statement *1149*
RETURN event *1943*
RETURN option
    ON ENDKEY phrase *841*
    ON ERROR phrase *843*
    ON QUIT phrase *846*
    ON STOP phrase *854*
    UNDO statement *1149*
RETURN PARAMETER option
    DEFINE PARAMETER statement *429*
RETURN statement *1012*
Return types
    FUNCTION statement *671*
    METHOD statement *808*
Return value
    RETURN statement *1012*
RETURN-INSERTED attribute *1780*
RETURN-TO-START-DIR option
    SYSTEM-DIALOG GET-DIR statement *1109*
    SYSTEM-DIALOG GET-FILE statement *1110*
RETURN-VALUE attribute *1780*
RETURN-VALUE function *1015*
RETURN-VALUE-DATA-TYPE attribute *1781*
RETURN-VALUE-DLL-TYPE attribute *1782*
ReturnExtent property *2215*
RETURNS option
    FUNCTION statement *671*
ReturnType property *2215*
ReturnTypeName property *2215*
ReturnValue property *2216*
REVERSE-FROM option
    QUERY-TUNING phrase *967*
REVERT option
    ON statement *848*
RGB-VALUE function *1015*
RIGHT event *1942*
RIGHT-ALIGNED option
    AT phrase *131*
RIGHT-MOUSE-CLICK event *1946*
RIGHT-MOUSE-DBLCLICK event *1946*
RIGHT-MOUSE-DOWN event *1946*
RIGHT-MOUSE-UP event *1946*
RIGHT-TRIM function *1016*
ROLE attribute *1782*
ROLES attribute *1783*
ROUND function *1018*
ROUNDED attribute *1783*
ROUNDED option
    DEFINE RECTANGLE statement *465*

Routine-level blocks
error redirection 1018
ROUTINE-LEVEL ON ERROR UNDO, THROW statement 1018
ROW attribute 1784
ROW option
AT phrase 131
CHOOSE statement 168
frame phrase 647
PUT CURSOR statement 944
PUT SCREEN statement 946
ROW-CREATE event 1957
ROW-DELETE event 1958
ROW-DISPLAY event 1949
ROW-ENTRY event 1949
ROW-HEIGHT-CHARS attribute 1784
ROW-HEIGHT-CHARS option
DEFINE BROWSE statement 366
ROW-HEIGHT-PIXELS attribute 1784
ROW-HEIGHT-PIXELS option
DEFINE BROWSE statement 366
ROW-LEAVE event 1949
Row-level events 1957
ROW-MARKERS attribute 1786
ROW-OF option
AT phrase 131
ROW-RESIZABLE attribute 1786
ROW-STATE attribute 1785
ROW-STATE function 1020
ROW-UPDATE event 1958
ROWID attribute 1785
ROWID data type 319
ROWID function 1021
RowModified property 2216
RowsAdded property 2216
RULE menu items 1860
RULE option
DEFINE MENU statement 423
DEFINE SUB-MENU statement 471
Rules
PAGE-TOP and PAGE-BOTTOM frames 647
RUN statement 1023
RUN STORED-PROCEDURE statement 1041
RUN SUPER statement 1044
RUN-PROCEDURE option
SUBSCRIBE statement 1096
Run-time parameters
RUN statement 1023
RVV option
COLOR phrase 210

S

SAVE CACHE statement 1047
SAVE method 1786
SAVE option
COMPILE statement 222
SAVE RESULT IN option
BUFFER-COMPARE statement 144
SAVE-AS option
SYSTEM-DIALOG GET-FILE statement 1110
SAVE-FILE method 1788
SAVE-ROW-CHANGES method 1789
SAVE-WHERE-STRING attribute 1790
SAX-attributes object handle
attributes 1318
methods 1318
SAX-PARSE method 1790
SAX-PARSE-FIRST method 1791
SAX-PARSE-NEXT method 1792
SAX-reader object handle
attributes 1319
methods 1319
SAX-writer object handle
attributes 1320
methods 1320
Scalar data types 320
Schema cache
saving 1047
SCHEMA-CHANGE attribute 1792
SCHEMA-LOCATION attribute 1793
SCHEMA-MARSHAL attribute 1793
SCHEMA-PATH attribute 1794
SCREEN-IO option
frame phrase 647
SCREEN-LINES attribute 1795
SCREEN-LINES function 1049
SCREEN-VALUE attribute 1795
SCROLL option
frame phrase 647
SCROLL statement 1049
SCROLL-BARS attribute 1796
SCROLL-HORIZONTAL event 1949
SCROLL-NOTIFY event 1950
SCROLL-TO-CURRENT-ROW method 1796
SCROLL-TO-ITEM method 1796
SCROLL-TO-SELECTED-ROW method 1797
SCROLL-VERTICAL event 1950
SCROLLABLE attribute 1797
SCROLLABLE option
frame phrase 647
SCROLLBAR-HORIZONTAL attribute 1798
SCROLLBAR-HORIZONTAL option
EDITOR phrase 564
SELECTION-LIST phrase 1061
SCROLLBAR-VERTICAL attribute 1798
SCROLLBAR-VERTICAL option
DEFINE BROWSE statement 366
EDITOR phrase 564
SELECTION-LIST phrase 1061

SCROLLING option
DEFINE QUERY statement *460*
SDBNAME function *1055*
SEAL method *1798*
SEAL-TIMESTAMP attribute *1800*
SEARCH function *1056*
SEARCH method *1800*
Second-level FILL events *1957*
SECTION option
GET-KEY-VALUE statement *696*
PUT-KEY-VALUE statement *958*
SECURITY-POLICY system handle
attributes *1321*
methods *1321*
SEEK function *1058*
SEEK statement *1059*
SELECT-ALL method *1801*
SELECT-FOCUSED-ROW method *1802*
SELECT-NEXT-ROW method *1803*
SELECT-PREV-ROW method *1803*
SELECT-ROW method *1803*
SELECTABLE attribute *1804*
SELECTED attribute *1804*
SELECTION event *1952*
SELECTION-END attribute *1805*
SELECTION-LIST phrase
VIEW-AS phrase *1180*
SELECTION-LIST widget
attributes *1244*
dynamic *299*
events *1244*
methods *1244*
SELECTION-START attribute *1805*
SELECTION-TEXT attribute *1805*
SELF system handle
attributes *1323*
Semicolon (;)
punctuation *63*
special character *63*
Send-sql-statement procedures
closing *207*
running *1041*
SENSITIVE attribute *1806*
SEPARATE-CONNECTION option
QUERY-TUNING phrase *967*
SEPARATOR-FGCOLOR attribute *1807*
SEPARATORS attribute *1806*
SEPARATORS option
DEFINE BROWSE statement *366*
SEQUENCE preprocessor name *74*
Sequences
dynamic *558*
static *830*
SERIALIZABLE option
CLASS statement *188*
DEFINE DATASET statement *393*
SERIALIZABLE option *(continued)*
DEFINE PROPERTY statement *442*
DEFINE TEMP-TABLE statement *477*
DEFINE VARIABLE statement *490*
VAR statement *1174*
SERIALIZE-HIDDEN attribute *1807*
SERIALIZE-HIDDEN option
DEFINE DATASET Statement *393*
DEFINE TEMP-TABLE statement *477*
SERIALIZE-NAME attribute *1807*
SERIALIZE-NAME option
DEFINE BUFFER statement *380*
DEFINE DATASET statement *393*
DEFINE PROPERTY statement *442*
DEFINE VARIABLE statement *490*
SERIALIZE-NAME option (field)
DEFINE TEMP-TABLE statement *477*
SERIALIZE-NAME option (temp-table)
DEFINE TEMP-TABLE statement *477*
SERIALIZE-ROW( ) method *1808*
Serialize( ) method (BinarySerializer) *2217*
Serialize( ) method (JsonSerializer) *2218*
SERVER attribute *1811*
Server object handle
attributes *1325*
methods *1325*
Server socket object handle
attributes *1326*
events *1326*
methods *1326*
SERVER-CONNECTION-BOUND attribute *1812*
SERVER-CONNECTION-BOUND-REQUEST attribute *1812*
SERVER-CONNECTION-CONTEXT attribute *1813*
SERVER-CONNECTION-ID attribute *1813*
SERVER-OPERATING-MODE attribute *1814*
SESSION system handle
attributes *1327*
methods *1327*
SESSION-END attribute *1815*
SESSION-ID attribute *1815*
SessionId property (OERequestInfo) *2220*
SET option
DEFINE PROPERTY statement *442*
MESSAGE statement *799*
RUN statement *1023*
WAIT-FOR statement *1190*
SET statement *1064*
SET-ACTOR method *1815*
SET-APPL-CONTEXT method *1816*
SET-ATTRIBUTE method *1817*
SET-ATTRIBUTE-NODE method *1818*
SET-BLUE-VALUE method *1818*
SET-BREAK method *1819*
SET-BUFFERS method *1820*
SET-BYTE-ORDER statement *1071*

SET-CALLBACK method 1821
SET-CALLBACK-PROCEDURE method 1822
SET-CLIENT method 1823
SET-COMMIT method 1826
SET-CONNECT-PROCEDURE method 1826
SET-CONTENTS option
SYSTEM-HELP statement 1115
SET-CURRENT-VALUE keyword value
DBRESTRICTIONS function 347
SET-DB-CLIENT function 1072
SET-DYNAMIC method 1827
SET-EFFECTIVE-TENANT function 1076
SET-GREEN-VALUE method 1827
SET-INPUT-SOURCE method 1828
SET-MUST-UNDERSTAND method 1829
SET-NODE method 1829
SET-NUMERIC-FORMAT method 1830
SET-OUTPUT-DESTINATION method 1831
SET-PARAMETER method (Handle) 1832
SET-POINTER-VALUE statement 1077
SET-PROPERTY method 1834
SET-READ-RESPONSE-PROCEDURE method 1835
SET-RED-VALUE method 1836
SET-REPOSITIONED-ROW method 1836
SET-RGB-VALUE method 1837
SET-ROLE method 1837
SET-ROLLBACK method 1838
SET-SAFE-USER method 1838
SET-SELECTION method 1839
SET-SERIALIZED method 1840
SET-SIZE statement 1078
SET-SOCKET-OPTION method 1841
SET-SORT-ARROW method 1842
SET-WAIT-STATE method 1844
Set( ) method (JsonArray) 2220
Set( ) method (JsonObject) 2222
Set( ) method (Reflect) 2224
SetByteOrder( ) method 2225
SetClientPrincipal( ) method 2226
SetFields( ) method 2227
SetFlag( ) method 2229
SetNull( ) method (JsonArray) 2230
SetNull( ) method (JsonObject) 2231
SetNumber( ) method (JsonArray) 2231
SetNumber( ) method (JsonObject) 2232
SetParameter( ) method (Class) 2233
SetPropertyValue( ) method 2235
SetterAccessMode property 2237
SETUSERID function 1080
SETUSERID keyword value
DBRESTRICTIONS function 347
Severity property 2237
SHA1-DIGEST function 1082
SHADOW-WINDOW widget
attributes 1246
events 1246
SHADOW-WINDOW widget *(continued)*
methods 1246
SHARE-LOCK option
CAN-FIND function 155
DEFINE BROWSE statement 366
FIND statement 609
FIND-BY-ROWID method 1541
GET statement 682
GET-CURRENT method 1570
GET-FIRST method 1574
GET-LAST method 1577
GET-NEXT method 1580
GET-PREV method 1582
Record phrase 985
SHARED FRAME option
DEFINE FRAME statement 410
Shared library routines
PROCEDURE statement 919
SHARED option
DEFINE BROWSE statement 366
DEFINE BUFFER statement 380
DEFINE DATASET statement 393
DEFINE MENU statement 423
DEFINE TEMP-TABLE statement 477
SHARED QUERY option
DEFINE QUERY statement 460
SHARED STREAM option
DEFINE STREAM statement 469
SHARED VARIABLE option
DEFINE VARIABLE statement 490
SHARED WORK-TABLE option
DEFINE WORK-TABLE statement 504
SHARED WORKFILE option
DEFINE WORK-TABLE statement 504
SHORT parameter data type 429
SHOW-IN-TASKBAR attribute 1844
SHOW-STATS statement 1083
SIDE-LABEL-HANDLE attribute 1845
SIDE-LABELS attribute 1845
SIDE-LABELS option
Frame phrase 647
SIGNATURE option
DEFINE EVENT statement 404
SIGNATURE-VALUE attribute 1846
SILENT option
DOS statement 543
OS-COMMAND statement 866
UNIX statement 1152
SIMPLE option
COMBO-BOX phrase 215
SINGLE option
DEFINE BROWSE statement 366
SELECTION-LIST phrase 1061
Single quote (')
special character 66
Single-line comments (//) 85

SINGLE-RUN attribute *1846*
SINGLE-RUN option
  RUN statement *1023*
SINGLETON attribute *1846*
SINGLETON option
  RUN statement *1023*
SIZE option
  SIZE phrase *1084*
Size phrase
  RADIO-SET phrase *975*
SIZE phrase
  COMBO-BOX phrase *215*
  DEFINE BROWSE statement *366*
  DEFINE BUTTON statement *386*
  DEFINE IMAGE statement *419*
  DEFINE RECTANGLE statement *465*
  EDITOR phrase *564*
  Frame phrase *647*
  SELECTION-LIST phrase *1061*
  SLIDER phrase *1085*
SIZE-CHARS option
  SIZE phrase *1084*
SIZE-PIXELS option
  SIZE phrase *1084*
SKIP menu items *1860*
SKIP option
  DEFINE FRAME statement *410*
  DEFINE MENU statement *423*
  DEFINE SUB-MENU statement *471*
  DISPLAY statement *531*
  ENABLE statement *568*
  FORM statement *632*
  MESSAGE statement *799*
  PROMPT-FOR statement *929*
  PUT statement *948*
  SET statement *1064*
  UPDATE statement *1157*
SKIP-DELETED-RECORD attribute *1846*
SkipBytes( ) method (FileInputStream) *2237*
SkipBytes( ) method (InputStream) *2238*
SkipBytes( ) method (MemoryInputStream) *2238*
Slash (/)
  special character *66*
SLIDER phrase
  VIEW-AS phrase *1180*
SLIDER widget
  attributes *1247*
  dynamic *299*
  events *1247*
  methods *1247*
SMALL-ICON attribute *1847*
SMALL-TITLE attribute *1847*
SOAP-fault object handle
  attributes *1332*
SOAP-FAULT-ACTOR attribute *1847*
SOAP-FAULT-CODE attribute *1847*
SOAP-FAULT-DETAIL attribute *1848*
SOAP-fault-detail object handle
  attributes *1333*
  methods *1333*
SOAP-FAULT-MISUNDERSTOOD-HEADER attribute *1848*
SOAP-FAULT-NODE attribute *1848*
SOAP-FAULT-ROLE attribute *1848*
SOAP-FAULT-STRING attribute *1849*
SOAP-FAULT-SUBCODE attribute *1849*
SOAP-header object handle
  attributes *1334*
  methods *1334*
SOAP-header-entryref object handle *1335*
SOAP-VERSION attribute *1849*
SoapFault property *2239*
Socket events *1955*
Socket object handle
  attributes *1336*
  events *1336*
  methods *1336*
Sockets
  CREATE SERVER-SOCKET statement *295*
  CREATE SOCKET statement *297*
  Server socket object *1326*
  Socket object handle *1336*
SORT attribute *1850*
SORT option
  COMBO-BOX phrase *215*
  SELECTION-LIST phrase *1061*
SORT-ASCENDING attribute *1850*
SORT-NUMBER attribute *1850*
Sorted property *2239*
SortRequest event *2282*
Source buffer phrase
  DEFINE DATASET statement *401*
SOURCE option
  DDE SEND statement *361*
  INPUT FROM statement *727*
  INPUT THROUGH statement *733*
  INPUT-OUTPUT THROUGH statement *738*
  OUTPUT THROUGH statement *876*
  OUTPUT TO statement *880*
SOURCE-PROCEDURE system handle
  attributes *1337*
  methods *1337*
SPACE option
  DEFINE FRAME statement *410*
  DISPLAY statement *531*
  ENABLE statement *568*
  FORM statement *632*
  PROMPT-FOR statement *929*
  PUT statement *948*
  SET statement *1064*
  UPDATE statement *1157*

SQL statements
DataServer applications 917
SQRT function 1088
SSL-SERVER-NAME attribute 1851
SSL-SERVER-NAME function 1089
Stack
Super procedure 1380
STANDALONE attribute 1851
START option
DDE ADVISE statement 352
START-BOX-SELECTION event 1954
START-DOCUMENT method 1851
START-ELEMENT method 1852
START-MOVE event 1953
START-RESIZE event 1953
START-ROW-RESIZE event 1953
START-SEARCH event 1950
STARTING AT option
COPY-LOB statement 264
STARTUP-PARAMETERS attribute 1853
StatBaseId property 2239
STATE-DETAIL attribute 1855
STATIC option
CONSTRUCTOR statement 260
DEFINE BUFFER statement 380
DEFINE DATA-SOURCE statement 401
DEFINE DATASET statement 393
DEFINE EVENT statement 404
DEFINE PROPERTY statement 442
DEFINE QUERY statement 460
DEFINE TEMP-TABLE statement 477
DEFINE VARIABLE statement 490
METHOD statement 808
VAR statement 1174
Static type-name syntax 1140
STATISTICS attribute 1856
STATUS statement 1089
STATUS-AREA attribute 1856
STATUS-AREA-FONT attribute 1856
STDCALL option
PROCEDURE statement 919
STOP attribute 1857
STOP conditions
RUN statement 1023
STOP option
DDE ADVISE statement 352
STOP statement 1091
STOP-AFTER phrase
DO statement 537, 620
REPEAT statement 1001
STOP-OBJECT attribute 1857
STOP-PARSING method 1857
STOPPED attribute 1858
Stored procedures
DISPLAY statement 531
INPUT option 1041
Stored procedures *(continued)*
INPUT-OUTPUT option 1041
LOAD-RESULT-INTO option 1041
NO-ERROR option 1041
OUTPUT option 1041
proc-text field name
DISPLAY statement 531
proc-text-buffer name
DEFINE BUFFER statement 380
FOR statement 620
retrieving return status 918
returning status values 207
running 1023
send-sql-statement
closing 207
running 1041
STREAM attribute 1858
Stream object handle
attributes 1340
example 1340
syntax 1340
STREAM option
DEFINE STREAM statement 469
DISPLAY statement 531
DOWN statement 544
EXPORT statement 594
frame phrase 647
HIDE statement 710
IMPORT statement 717
INPUT CLOSE statement 726
INPUT FROM statement 727
INPUT THROUGH statement 733
INPUT-OUTPUT CLOSE statement 737
INPUT-OUTPUT THROUGH statement 738
OUTPUT CLOSE statement 875
OUTPUT THROUGH statement 876
OUTPUT TO statement 880
PAGE statement 890
PROMPT-FOR statement 929
PUT statement 948
READKEY statement 982
SEEK statement 1059
SET statement 1064
UNDERLINE statement 1148
UP statement 1156
VIEW statement 1178
STREAM-HANDLE option
DISPLAY statement 531
DOWN statement 544
EXPORT statement 594
Frame phrase 647
HIDE statement 710
IMPORT statement 717
INPUT CLOSE statement 726
INPUT FROM statement 727
INPUT THROUGH statement 733

STREAM-HANDLE option *(continued)*
INPUT-OUTPUT CLOSE statement *737*
INPUT-OUTPUT THROUGH statement *738*
LINE-COUNTER function *779*
OUTPUT CLOSE statement *875*
OUTPUT THROUGH statement *876*
OUTPUT TO statement *880*
PAGE statement *890*
PAGE-NUMBER function *891*
PAGE-SIZE function *891*
PROMPT-FOR statement *929*
PUT statement *948*
READKEY statement *982*
SEEK function *1058*
SEEK statement *1059*
SET statement *1064*
UNDERLINE statement *1148*
UP statement *1156*
VIEW statement *1178*
STREAM-IO option
COMPILE statement *222*
frame phrase *647*
STRETCH-TO-FIT attribute *1858*
STRETCH-TO-FIT option
DEFINE IMAGE statement *419*
STRICT attribute *1859*
STRICT-ENTITY-RESOLUTION attribute *1859*
STRING function *1094*
String property *2240*
STRING-VALUE method *1859*
STRING-XREF option
COMPILE statement *222*
SUB-AVERAGE option
aggregate phrase *111*
SUB-COUNT option
aggregate phrase *111*
SUB-MAXIMUM option
aggregate phrase *111*
SUB-MENU option
DEFINE MENU statement *423*
DEFINE SUB-MENU statement *471*
Widget phrase *1201*
SUB-MENU widget
attributes *1249*
dynamic *299*
events *1249*
methods *1249*
SUB-MENU-HELP option
DEFINE SUB-MENU statement *471*
SUB-MINIMUM option
aggregate phrase *111*
SUB-TOTAL option
aggregate phrase *111*
Subclass
defined *315*
SUBSCRIBE statement *1096*
Subscribe( ) method *2240*, *2283*
SUBSTITUTE function *1098*
SUBSTRING function *1099*
SUBSTRING statement *1100*
Subtract( ) method *2240*
Subtraction assignment operator (-=) *98*
Subtraction operator (-) *94*
SUBTYPE attribute *1860*
Success property *2240*
SUPER function *1103*
Super procedures *1103*, *1105*, *1380*, *1766*
SUPER statement *1104*
SUPER system reference *1105*
SUPER-PROCEDURES attribute *1861*
SuperClass property *2241*
SUPPRESS-NAMESPACE-PROCESSING attribute *1861*
SUPPRESS-WARNINGS attribute *1861*
SUPPRESS-WARNINGS-LIST attribute *1861*
SYMMETRIC-ENCRYPTION-ALGORITHM attribute *1862*
SYMMETRIC-ENCRYPTION-IV attribute *1862*
SYMMETRIC-ENCRYPTION-KEY attribute *1862*
SYMMETRIC-SUPPORT attribute *1863*
SYNCHRONIZE event *1959*
SYNCHRONIZE method *1864*
SyncToSelectedRow property *2241*
SYSTEM-ALERT-BOXES attribute *1864*
SYSTEM-DIALOG COLOR statement *1106*
SYSTEM-DIALOG FONT statement *1107*
SYSTEM-DIALOG GET-DIR statement *1109*
SYSTEM-DIALOG GET-FILE statement *1110*
SYSTEM-DIALOG PRINTER-SETUP statement *1114*
SYSTEM-HELP statement *1115*
SYSTEM-ID attribute *1865*
System.Exception class
events *2049*
methods *2049*
properties *2049*

T

T option
Character-string literal (" ") *67*
TAB event *1943*
TAB-POSITION attribute *1865*
TAB-STOP attribute *1865*
TABLE attribute *1865*
TABLE option
DEFINE PARAMETER statement *429*
Parameter definition syntax *892*
Parameter passing syntax *898*
TABLE-CRC-LIST attribute *1866*
TABLE-HANDLE attribute *1866*
TABLE-HANDLE option
DEFINE PARAMETER statement *429*
Parameter definition syntax *892*

TABLE-HANDLE option *(continued)*
Parameter passing syntax *898*
TABLE-LIST attribute *1866*
TABLE-NUMBER attribute *1867*
TABLE-SCAN option
Record phrase *985*
TableName property *2242*
TableSchema property *2242*
TableStatId property *2243*
Tag property (ActiveX control) *1867*
Tag property (Progress.Data.BindingSource) *2243*
TARGET option
DDE GET statement *356*
DDE REQUEST statement *359*
INPUT FROM statement *727*
INPUT THROUGH statement *733*
INPUT-OUTPUT THROUGH statement *738*
OUTPUT THROUGH statement *876*
OUTPUT TO statement *880*
TARGET-PROCEDURE system handle
attributes *1340*
methods *1340*
TEMP-DIRECTORY attribute *1867*
Temp-table object
CREATE TEMP-TABLE statement *297*
Temp-table object handle
attributes *1343*
methods *1343*
TEMP-TABLE option
DEFINE BUFFER statement *380*
DEFINE PARAMETER statement *429*
DEFINE TEMP-TABLE statement *477*
TEMP-TABLE-PREPARE method *1868*
TempTableCount property *2243*
TempTablePeak property *2243*
TENANT-ID function *1120*
TENANT-ID method *1868*
TENANT-NAME function *1121*
TENANT-NAME method *1869*
TENANT-NAME-TO-ID function *1121*
TERMINAL function *1122*
TERMINAL option
INPUT FROM statement *727*
OUTPUT TO statement *880*
TERMINAL statement *1123*
Text field
Key actions *1064*
TEXT option
ENABLE statement *568*
PROMPT-FOR statement *929*
SET statement *1064*
SYSTEM-HELP statement *1115*
UPDATE statement *1157*
VIEW-AS phrase *1180*
Text segments
generation *222*
TEXT widget
attributes *1250*
dynamic *299*
events *1250*
methods *1250*
TEXT-SEG-GROW option
COMPILE statement *222*
TEXT-SELECTED attribute *1870*
THEN function *712*
THEN option
BUFFER-COMPARE statement *144*
CASE statement *161*
THEN statement *713*
THIS-OBJECT statement *1124*
THIS-OBJECT system reference *1124*
THIS-PROCEDURE system handle
attributes *1345*
methods *1345*
THREAD-SAFE attribute *1870*
THREAD-SAFE option
PROCEDURE statement *919*
ThreadId property *2244*
THREE-D attribute *1871*
THREE-D option
frame phrase *647*
THROW option
BLOCK-LEVEL ON ERROR UNDO, THROW statement *140*
ON ERROR phrase *843*
ROUTINE-LEVEL ON ERROR UNDO, THROW statement *1018*
UNDO statement *1149*
TIC-MARKS attribute *1871*
TIC-MARKS option
SLIDER phrase *1085*
Tilde (~)
special character *65*
Time
elapsed *592*
TIME function *1126*
TIME option
DDE ADVISE statement *352*
DDE EXECUTE statement *354*
DDE GET statement *356*
DDE REQUEST statement *359*
DDE SEND statement *361*
TIME-SOURCE attribute *1872*
TIMEZONE attribute *1872*
TIMEZONE function *1127*
TITLE attribute *1873*
TITLE option
DEFINE MENU statement *423*
MESSAGE statement *799*
SYSTEM-DIALOG GET-DIR statement *1109*
SYSTEM-DIALOG GET-FILE statement *1110*

TITLE phrase
  DEFINE BROWSE statement *366*
  frame phrase *647*
TITLE-BGCOLOR attribute *1873*
TITLE-DCOLOR attribute *1874*
TITLE-FGCOLOR attribute *1874*
TITLE-FONT attribute *1874*
TO option
  APPLY statement *119*
  BUFFER-COMPARE statement *144*
  BUFFER-COPY statement *146*
  DEFINE FRAME statement *410*
  DO statement *537*
  ENABLE statement *568*
  FOR statement *620*
  FORM statement *632*
  format phrase *638*
  PROMPT-FOR statement *929*
  PUT statement *948*
  SEEK statement *1059*
  SET statement *1064*
  SUBSCRIBE statement *1096*
  UPDATE statement *1157*
TO RECID NO-ERROR option
  REPOSITION statement *1008*
TO RECID option
  REPOSITION statement *1008*
TO ROW option
  REPOSITION statement *1008*
TO ROWID NO-ERROR option
  REPOSITION statement *1008*
TO ROWID option
  REPOSITION statement *1008*
TO-ROWID function *1128*
TODAY function *1127*
TOGGLE-BOX attribute *1875*
TOGGLE-BOX option
  DEFINE MENU statement *423*
  DEFINE SUB-MENU statement *471*
  VIEW-AS phrase *1180*
TOGGLE-BOX widget
  attributes *1251*
  dynamic *299*
  events *1251*
  methods *1251*
ToggleFlag( ) method *2244*
ToObject( ) method *2245*
TOOLTIP attribute *1875*
TOOLTIP option
  COMBO-BOX phrase *215*
  DEFINE BUTTON statement *386*
  DEFINE IMAGE statement *419*
  DEFINE RECTANGLE statement *465*
  EDITOR phrase *564*
  RADIO-SET phrase *975*
  SELECTION-LIST phrase *1061*
TOOLTIP option *(continued)*
  SLIDER phrase *1085*
  VIEW-AS phrase *1180*
TOOLTIPS attribute *1875*
Top property *1875*
TOP-NAV-QUERY attribute *1876*
TOP-ONLY attribute *1876*
TOP-ONLY option
  frame phrase *647*
TOPIC option
  DDE INITIATE statement *358*
ToString( ) method (Object) *2246*
ToString( ) method (Reflect) *2248*
TOTAL option
  aggregate phrase *111*
TRACE-FILTER attribute *1877*
TRACING attribute *1878*
TRACKING-CHANGES attribute *1878*
TRANS-INIT-PROCEDURE attribute *1880*
TRANSACTION attribute *1879*
TRANSACTION DISTINCT option
  RUN statement *1023*
TRANSACTION function *1130*
Transaction object handle
  attributes *1348*
  methods *1348*
TRANSACTION option
  DO statement *537*
  FOR statement *620*
  REPEAT statement *1001*
TRANSACTION-MODE AUTOMATIC statement *1130*
TransId property *2248*
TRANSPARENT attribute *1879*
TRANSPARENT option
  DEFINE IMAGE statement *419*
Trigger phrase
  CREATE BROWSE statement *278*
  CREATE widget statement *299*
  DEFINE BUTTON statement *386*
  DEFINE MENU statement *423*
  DEFINE RECTANGLE statement *465*
  DEFINE SUB-MENU statement *471*
  DEFINE VARIABLE statement *490*
TRIGGER PROCEDURE statement *1133*
Triggers
  current *1323*
  low-level keyboard events *1939*
  user-interface *1323*
TRIM function *1137*
TRIM option
  COPY-LOB statement *264*
TRUNCATE function *1139*
TTCODEPAGE option
  DEFINE TEMP-TABLE statement *477*
TYPE attribute *1880*
Type names *1140*

Type-name syntax
static *1140*
TYPE-OF function *1146*
TypeName property *2248*

U

U option
Character-string literal (" ") *67*
U1 through U10 events *1954*
Unary negative operator *94*
Unary positive operator (+) *86*
UNBOX function *1146*
Unboxing
defined *325*
UNBUFFERED option
INPUT FROM statement *727*
INPUT THROUGH statement *733*
INPUT-OUTPUT THROUGH statement *738*
OUTPUT THROUGH statement *876*
OUTPUT TO statement *880*
UNDERLINE option
COLOR phrase *210*
UNDERLINE statement *1148*
UNDO attribute *1881*
UNDO option
ON QUIT phrase *846*
UNDO statement *1149*
UNDO-THROW-SCOPE attribute *1881*
UNFORMATTED option
IMPORT statement *717*
PUT statement *948*
UNIQUE option
DEFINE TEMP-TABLE statement *477*
UNIQUE-ID attribute
x-noderef handle uniqueness *1351*
UNIQUE-MATCH attribute *1882*
UNIQUE-MATCH option
COMBO-BOX phrase *215*
Universal Key function events *1940*
UNIX statement *1152*
UnknownUser property *2249*
UNLESS-HIDDEN option
DISABLE statement *525*
DISPLAY statement *531*
PROMPT-FOR statement *929*
SET statement *1064*
UPDATE statement *1157*
UNLESS-HIDDEN Option
DISABLE statement *568*
UNLOAD statement *1154*
UnsetFlag( ) method *2249*
UNSUBSCRIBE statement *1155*
Unsubscribe( ) method *2284*
UP event *1942*
UP option
SCROLL statement *1049*
UP statement *1156*
UPDATE option
MESSAGE statement *799*
SYSTEM-DIALOG COLOR statement *1106*
SYSTEM-DIALOG FONT statement *1107*
SYSTEM-DIALOG GET-FILE statement *1110*
SYSTEM-DIALOG PRINTER-SETUP statement *1114*
UPDATE statement *1157*
UPDATE-ATTRIBUTE method *1882*
URL attribute *1884*
URL-DECODE method *1884*
URL-ENCODE method *1884*
URL-PASSWORD attribute *1884*
URL-USERID attribute *1884*
USE statement *1164*
USE-DICT-EXPS option
frame phrase *647*
USE-FILENAME option
SYSTEM-DIALOG GET-FILE statement *1110*
USE-INDEX option
CAN-FIND function *155*
DEFINE TEMP-TABLE statement *477*
FIND statement *609*
Record phrase *985*
USE-REVVIDEO option
frame phrase *647*
V6FRAME option
COMPILE statement *222*
USE-TEXT option
frame phrase *647*
USE-UNDERLINE option
frame phrase *647*
V6FRAME option
COMPILE statement *222*
USE-WIDGET-POOL option
CLASS statement *187*
UseFullFieldNames property *2250*
User property *2251*
USER-DATA() method *1885*
User-defined functions *671*
User-defined types
See also Data types
class *185*
interface *747*
name syntax *1140*
See also Data types
USER-ID attribute *1885*
USERID function *1165*
UserIndexStatId property *2251*
UserIOId property *2252*
UserTableStatId property *2252*
USING option
BUFFER-COMPARE statement *144*

USING option *(continued)*
BUFFER-COPY statement *146*
CAN-FIND function *155*
FIND statement *609*
Record phrase *985*
USING RECID option
CREATE statement *268*
INSERT statement *742*
USING ROWID option
CREATE statement *268*
INSERT statement *742*
USING statement *1167*

V

V6DISPLAY attribute *1886*
V6FRAME option
COMPILE statement *222*
frame phrase *647*
VALID-EVENT function *1171*
VALID-HANDLE function *1171*
VALID-OBJECT function *1173*
VALIDATE method *1887*
VALIDATE option
DEFINE BROWSE statement *366*
DEFINE TEMP-TABLE statement *477*
DEFINE WORK-TABLE statement *504*
DELETE statement *510*
format phrase *638*
VALIDATE statement *1173*
VALIDATE-DOMAIN-ACCESS-CODE method *1888*
VALIDATE-EXPRESSION attribute *1888*
VALIDATE-MESSAGE attribute *1888*
VALIDATE-SEAL method *1889*
VALIDATE-XML attribute *1890*
Validating
handles *1173*
message authentication codes (MAC) *1889*
object references *1173*
procedure handles *1171*
records *1173*
widget events *1171*
XML *1890*
VALIDATION-ENABLED attribute *1890*
VALUE attribute *1890*
VALUE option
COLOR phrase
example *210*
COMPILE statement *222*
CONNECT statement *253*
CREATE widget statement *299*
DELETE ALIAS statement *513*
DISCONNECT statement *530*
DOS statement *543*
GET-KEY-VALUE statement *696*
INPUT FROM statement *727*
VALUE option *(continued)*
INPUT THROUGH statement *733*
INPUT-OUTPUT THROUGH statement *738*
OS-APPEND statement *864*
OS-COMMAND statement *866*
OS-COPY statement *867*
OS-CREATE-DIR statement *868*
OS-DELETE statement *869*
OS-RENAME statement *874*
OUTPUT THROUGH statement *876*
OUTPUT TO statement *880*
PUT-KEY-VALUE statement *958*
RUN statement *1023*
RUN VALUE example *1023*
SAVE CACHE statement *1047*
UNIX statement *1152*
VALUE-CHANGED event *1950*
VAR statement *1174*
VARIABLE option
DEFINE VARIABLE statement *490*
VERSION attribute *1891*
VERSION function *351*, *940*
See also DBVERSION function
VersionInfo property *2252*
VERTICAL option
RADIO-SET phrase *975*
SLIDER phrase *1085*
VIEW statement *1178*
VIEW-AS ALERT-BOX option
MESSAGE statement *799*
VIEW-AS attribute *1891*
VIEW-AS DIALOG-BOX option
example *410*
frame phrase
DEFINE FRAME statement *410*
VIEW-AS phrase
DEFINE BROWSE statement *366*
DEFINE VARIABLE statement *490*
format phrase *638*
SET statement *1064*
VIEW-AS TEXT option
DEFINE FRAME statement *410*
ENABLE statement *568*
FORM statement *632*
PROMPT-FOR statement *929*
VIEW-FIRST-COLUMN-ON-REOPEN attribute *1892*
VIRTUAL-HEIGHT-CHARS attribute *1892*
VIRTUAL-HEIGHT-PIXELS attribute *1893*
VIRTUAL-WIDTH-CHARS attribute *1893*
VIRTUAL-WIDTH-PIXELS attribute *1893*
VISIBLE attribute *1893*
Visual Basic equivalents
CREATE automation object statement *274*
VOID option
DEFINE EVENT statement *404*
METHOD statement *808*

W

WAIT-FOR statement
  .NET and ABL *1190*
  ABL only *1186*
WARNING attribute *1895*
WC-ADMIN-APP attribute *1895*
WEB-CONTEXT system handle
  attributes *1349*
  methods *1349*
WEB-NOTIFY event *1937*
WEBSTREAM preprocessor name *74*
WEEKDAY function *1199*
WHEN option
  ASSIGN statement *126*
  BUFFER-COMPARE statement *144*
  CASE statement *161*
  DISABLE statement *525*
  DISPLAY statement *531*
  ENABLE statement *568*
  PROMPT-FOR statement *929*
  SET statement *1064*
  UPDATE statement *1157*
WHERE option
  CAN-FIND function *155*
  case sensitivity *985*
  CLOSE STORE-PROCEDURE statement *207*
  FIND statement *609*
  Record phrase *985*
WHERE-STRING attribute *1895*
WHILE option
  DO statement *537*
  FOR statement *620*
  REPEAT statement *1001*
Widget phrase
  browse columns *1201*
  field-level widgets *1201*
  HIDE statement *710*
  VIEW statement *1178*
WIDGET-ENTER attribute *1897*
WIDGET-HANDLE data type *319*
WIDGET-HANDLE function *1200*
Widget-Handle property *1896*
WIDGET-ID attribute *1897*
WIDGET-ID option
  DEFINE FRAME statement *410*
  FORM statement *632*
  format phrase *638*
  frame phrase *647*
WIDGET-LEAVE attribute *1898*
Widgets
  colors, fonts, and measurements *1358*
  dynamic *299*
  Reference *1207*
WIDTH option
  DEFINE BROWSE statement *366*
WIDTH option *(continued)*
  frame phrase *647*
Width property *1899*
WIDTH-CHARS attribute *1899*
WIDTH-PIXELS attribute *1900*
WINDOW attribute *1900*
Window families *1253*
WINDOW widget
  attributes *1253*
  dynamic *299*
  events *1253*
  methods *1253*
WINDOW-CLOSE event *1950*
WINDOW-MAXIMIZED event *1951*
WINDOW-MINIMIZED event *1951*
WINDOW-NAME option
  SYSTEM-HELP statement *1115*
WINDOW-RESIZED event *1951*
WINDOW-RESTORED event *1951*
WINDOW-STATE attribute *1900*
WINDOW-SYSTEM attribute *1901*
WINDOW-SYSTEM preprocessor name *74*
WITH BROWSE option
  DISPLAY statement *531*
Word indexes
  CONTAINS operator *985*
  temporary tables *477*
WORD-INDEX option
  DEFINE TEMP-TABLE statement *477*
WORD-WRAP attribute *1902*
WORK-AREA-HEIGHT-PIXELS attribute *1902*
WORK-AREA-WIDTH-PIXELS attribute *1902*
WORK-AREA-X attribute *1903*
WORK-AREA-Y attribute *1903*
WORK-TABLE option
  DEFINE WORK-TABLE statement *504*
WORKFILE option
  DEFINE WORK-TABLE statement *504*
Working directory *1203*
WRITE method (Socket) *1903*
WRITE-CDATA method *1904*
WRITE-CHARACTERS method *1905*
WRITE-COMMENT method *1905*
WRITE-DATA-ELEMENT method *1906*
WRITE-DATA() method *1906*
WRITE-EMPTY-ELEMENT method *1908*
WRITE-ENTITY-REF method *1909*
WRITE-EXTERNAL-DTD method *1909*
WRITE-FRAGMENT method *1910*
WRITE-JSON method *1911*
WRITE-MESSAGE method *1917*
WRITE-PROCESSING-INSTRUCTION method *1918*
WRITE-STATUS attribute *1918*
WRITE-XML method *1919*
WRITE-XMLSCHEMA method *1925*
Write( ) method (JsonArray) *2254*

Write( ) method (JsonConstruct) *2256*
Write( ) method (JsonObject) *2258*
Write( ) method (MemoryOutputStream) *2253*
Write( ) method (OutputStream) *2260*
WriteByte( ) method *2261*
WriteDouble( ) method *2261*
WriteFile( ) method (JsonArray) *2262*
WriteFile( ) method (JsonConstruct) *2263*
WriteFile( ) method (JsonObject) *2263*
WriteFloat( ) method *2264*
WriteInt64( ) method *2265*
WriteLong( ) method *2265*
WriteShort( ) method *2265*
WriteStream( ) method (JsonArray) *2266*
WriteStream( ) method (JsonConstruct) *2267*
WriteStream( ) method (JsonObject) *2268*
WriteUnsignedLong( ) method *2268*
WriteUnsignedShort( ) method (MemoryOutputStream) *2269*

X

X attribute *1929*
X option
    AT phrase *131*
X-DOCUMENT attribute *1930*
X-document object handle
    attributes *1350*
    methods *1350*
X-noderef object handle
    attributes *1351*
    methods *1351*
X-OF option
    AT phrase *131*
XCODE option
    COMPILE statement *222*
XCODE-SESSION-KEY attribute *1930*
XML-DATA-TYPE attribute *1930*
XML-DATA-TYPE option
    DEFINE TEMP-TABLE statement *477*
XML-ENTITY-EXPANSION-LIMIT attribute *1931*
XML-NODE-NAME attribute *1931*
XML-NODE-NAME option
    DEFINE BUFFER statement *380*
    DEFINE DATASET statement *393*
XML-NODE-NAME option (field)
    DEFINE TEMP-TABLE statement *477*
XML-NODE-NAME option (temp-table)
    DEFINE TEMP-TABLE statement *477*
XML-NODE-TYPE attribute *1932*
XML-NODE-TYPE option
    DEFINE DATASET Statement *393*
    DEFINE TEMP-TABLE statement *477*
XML-SCHEMA-PATH attribute *1933*
XML-STRICT-ENTITY-RESOLUTION attribute *1934*
XML-SUPPRESS-NAMESPACE-PROCESSING attribute *1935*
XOR operator (bitwise) *1204*
Xor( ) method *2269*
XREF option
    COMPILE statement *222*
XREF-XML option
    COMPILE statement *222*

Y

Y attribute *1936*
Y option
    AT phrase *131*
Y-OF option
    AT phrase *131*
YEAR function *1205*
YEAR-OFFSET attribute *1936*