

Progress
ADM 2 Reference

© 2002 Progress Software Corporation. All rights reserved.

Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. This manual is also copyrighted and all rights are reserved. This manual may not, in whole or in part, be copied, photocopied, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Progress Software Corporation.

The information in this manual is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear in this document.

The references in this manual to specific platforms supported are subject to change.

Progress, Powered by Progress, Progress Fast Track, Progress Profiles, Partners in Progress, Partners en Progress, Progress en Partners, Progress in Progress, P.I.P., Progress Results, ProVision, ProCare, ProtoSpeed, SmartBeans, SpeedScript, and WebSpeed are registered trademarks of Progress Software Corporation in the U.S. and other countries. A Data Center of Your Very Own, Allegrix, Apptivity, AppsAlive, AppServer, ASPen, ASP-in-a-Box, Empowerment Center, Fathom, Future Proof, IntelliStream, OpenEdge, PeerDirect, POSSE, POSSENET, Progress Dynamics, Progress Software Developers Network, SectorAlliance, SmartObjects and WebClient are trademarks or service marks of Progress Software Corporation in the U.S. and other countries.

SonicMQ is a registered trademark of Sonic Software Corporation in the U.S. and other countries.

Vermont Views is a registered trademark of Vermont Creative Software in the U.S. and other countries.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Any other trademarks and/or service marks contained herein are the property of their respective owners.

Progress Software Corporation acknowledges the use of Raster Imaging Technology copyrighted by Snowbound Software 1993-1997, the IBM XML Parser for Java Edition, and software developed by the Apache Software Foundation (http://www.apache.org/).

© IBM Corporation 1998-1999. All rights reserved. U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Progress is a registered trademark of Progress Software Corporation and is used by IBM Corporation in the mark Progress/400 under license. Progress/400 AND 400® are trademarks of IBM Corporation and are used by Progress Software Corporation under license.

The implementation of the MD5 Message-Digest Algorithm used by Progress Software Corporation in certain of its products is derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm.

May 2002



Product Code: 4606
Item Number: 91697;V9.1D

Contents

Preface . . . . . **xxvii**
Purpose xxvii
Audience xxvii
Organization Of This Manual xxvii
How To Use This Manual xxviii
Typographical Conventions xxix
Syntax Notation xxx
Example Procedures xxxiv
Progress Messages xxxvii
Other Useful Documentation xxxix
Getting Started xxxix
Development Tools xl
Reporting Tools xli
4GL xlii
Database xliii
DataServers xliii
SQL-89/Open Access xliv
SQL-92 xliv
Deployment xlv
WebSpeed xlv
Reference xlvi

ADM2 SmartObject API Reference **1**
Paths To Source Files 2
Source-file Types 3
ADM Class Hierarchy 4
SmartObjects, Templates, and Super-procedure Hierarchies 5

SmartObjects and Their Files 7
Simple SmartObject 7
Simple SmartContainer 8
SmartBusinessObject 9
SmartB2BObject 10
SmartCombo 12
SmartConsumer 13
SmartDataBrowser 13
SmartDataField 14
SmartDataObject 15
SmartDataViewer 17
SmartDialog 18
SmartFilter 19
SmartFolder 20
SmartFrame 21
SmartLookup 21
SmartPanel 22
SmartProducer, SmartConsumer 23
SmartReceiver 25
SmartRouter 25
SmartSelect 26
SmartSender, SmartReceiver 27
SmartToolbar 27
SmartWindow 29
Reading and Writing Object Properties 30
get*propname* and set*propname* Functions 31
{get} and {set} Pseudo-functions 31
The Need For Two Different Syntaxes 32
List Of Public Properties By Name 33

Alphabetical Listing of API Routines 59
actionAccelerator (action.p) 60
actionAccessType (action.p) 60
actionCanRun (toolbar.p) 60
actionCaption (action.p) 61
actionCaption (toolbar.p) 61
actionCategory (action.p) 61
actionCategoryIsHidden (toolbar.p) 62
actionChecked (toolbar.p) 62
actionChildren (action.p) 63
actionControlType (action.p) 63
actionCreateEvent (action.p) 64
actionDescription (action.p) 64
actionDisabled (action.p) 64
actionEnableRule (action.p) 65

actionEnableRulex (action.p) . . . 65
actionGroups (action.p) . . . 65
actionHideRule (action.p) . . . 66
actionHideRuleX (action.p) . . . 66
actionImage (action.p) . . . 66
actionImageAlternate (action.p) . . . 67
actionImageAlternateRule (action.p) . . . 67
actionInitCode (action.p) . . . 67
actionIsMenu (action.p) . . . 68
actionIsParent (action.p) . . . 68
actionLabel (action.p) . . . 68
actionLabel (toolbar.p) . . . 69
actionLink (action.p) . . . 69
actionLogicalObjectName (action.p) . . . 69
actionName (action.p) . . . 70
actionOnChoose (action.p) . . . 70
actionParameter (action.p) . . . 70
actionParent (action.p) . . . 71
actionPhysicalObjectName (action.p) . . . 71
actionPublishCreate (toolbar.p) . . . 72
actionRefresh (action.p) . . . 72
actionRunAttribute (action.p) . . . 73
actionSecondImage (action.p) . . . 73
actionSecuredToken (action.p) . . . 73
actionSubstituteProperty (action.p) . . . 74
actionTarget (toolbar.p) . . . 74
actionTooltip (action.p) . . . 74
actionTooltip (toolbar.p) . . . 75
actionType (action.p) . . . 75
activeTarget (panel.p) . . . 76
addDataTarget (sbo.p) . . . 77
addLink (smart.p) . . . 78
addMessage (smart.p) . . . 80
addNavigationSource (sbo.p) . . . 81
addNode (treeview.p) . . . 82
addQueryWhere (query.p) . . . 83
addQueryWhere (sbo.p) . . . 84
addRecord (browser.p) . . . 84
addRecord (datavis.p) . . . 85
addRecord (viewer.p) . . . 86
addRow (data.p) . . . 87
addRow (sbo.p) . . . 88
adjustActions (toolbar.p) . . . 88
adjustTabOrder (smart.p) . . . 88
anyKey (combo.p) . . . 89

anyKey (lookup.p) 89
anyKey (select.p) 90
anyMessage (smart.p) 90
appendContainedObjects (sbo.p) 91
applyCellEntry (browser.p) 91
applyEntry (browser.p) 92
applyEntry (smart.p) 93
applyFilter (filter.p) 94
applyLayout (visual.p) 94
assignActionAccelerator (action.p) 95
assignActionAccessType (action.p) 95
assignActionAlternateImageRule (action.p) 96
assignActionCaption (action.p) 96
assignActionDescription (action.p) 97
assignActionDisableRule (action.p) 97
assignActionEnableRule (action.p) 98
assignActionHideRule (action.p) 98
assignActionImage (action.p) 99
assignActionLabel (action.p) 99
assignActionName (action.p) 100
assignActionOrder (action.p) 100
assignActionParent (action.p) 101
assignActionSecondImage (action.p) 101
assignActionTooltip (action.p) 102
assignAttribute (xml.p) 102
assignColumn (action.p) 103
assignColumnColumnLabel (dataextcols.p) 103
assignColumnFormat (dataextcols.p) 104
assignColumnFormat (filter.p) 104
assignColumnHelp (dataextcols.p) 105
assignColumnHelpId (filter.p) 105
assignColumnLabel (dataextcols.p) 106
assignColumnLabel (filter.p) 106
assignColumnOperatorStyle (filter.p) 107
assignColumnPrivateData (dataextcols.p) 107
assignColumnTooltip (filter.p) 108
assignColumnValExp (dataextcols.p) 108
assignColumnValMsg (dataextcols.p) 109
assignColumnWidth (filter.p) 109
assignContainedProperties (sbo.p) 110
assignCurrentMappedObject (sbo.p) 110
assignDBRow (query.p) 111
assignLinkProperty (smart.p) 111
assignMaxGuess (browser.p) 112
assignMaxGuess (sbo.p) 112

assignNodeValue (xml.p) . . . 113
assignPageProperty (containr.p) . . . 113
assignQuerySelection (query.p) . . . 114
assignQuerySelection (sbo.p) . . . 115
assignUnsubscribe (consumer.p) . . . 116
bandActions (toolbar.p) . . . 116
bandSubmenuLabel (toolbar.p) . . . 116
bandSubmenus (toolbar.p) . . . 117
batchServices (data.p) . . . 117
beginTransactionValidate (datalogic.i) . . . 118
bindServer (appserver.p) . . . 118
blankField (filter.p) . . . 118
blankFields (filter.p) . . . 119
blankFillIn (filter.p) . . . 119
browseHandler (select.p) . . . 120
bufferCompareDBToRO (query.p) . . . 120
bufferCopyDBToRO (query.p) . . . 121
bufferHandle (action.p) . . . 122
bufferWhereClause (query.p) . . . 122
buildAllMenus (toolbar.p) . . . 123
buildList (select.p) . . . 123
buildMenu (toolbar.p) . . . 123
calcWidth (browser.p) . . . 123
callOutParams (b2b.p) . . . 124
cancelNew (browser.p) . . . 124
cancelNew (sbo.p) . . . 124
cancelObject (containr.p) . . . 125
cancelObject (datavis.p) . . . 125
cancelRecord (browser.p) . . . 126
cancelRecord (datavis.p) . . . 126
cancelRecord (viewer.p) . . . 127
cancelRow (data.p) . . . 128
cancelRow (sbo.p) . . . 128
canFindAction (action.p) . . . 128
canFindCategory (action.p) . . . 129
canFindModRow (data.i) . . . 129
canNavigate (data.p) . . . 129
canNavigate (datavis.p) . . . 130
canNavigate (sbo.p) . . . 130
categoryActions (toolbar.p) . . . 131
categoryLink (action.p) . . . 131
changeCursor (smart.p) . . . 131
changePage (containr.p) . . . 132
characterValue (b2b.p) . . . 133
checkRule (panel.p) . . . 133

clearActionCache (action.p) 134
clearCache (action.p) 134
clearCache (toolbar.p) 134
clearLogicRows (logic.i) 134
clearToolbarCache (toolbar.p) 135
clientSendRows (data.p) 135
closeQuery (data.p) 136
closeQuery (query.p) 136
collectChanges (datavis.p) 137
columnColumnLabel (dataextcols.p) 138
columnColumnLabel (sbo.p) 138
columnDataType (dataextcols.p) 139
columnDataType (filter.p) 139
columnDataType (query.p) 140
columnDataType (sbo.p) 140
columnDbColumn (query.p) 141
columnDbColumn (sbo.p) 141
columnExtent (dataextcols.p) 141
columnExtent (sbo.p) 142
columnFilterTarget (filter.p) 142
columnFormat (dataextcols.p) 142
columnFormat (filter.p) 143
columnFormat (sbo.p) 143
columnHandle (dataextcols.p) 144
columnHelp (dataextcols.p) 144
columnHelp (sbo.p) 144
columnHelpId (filter.p) 145
columnInitial (dataextcols.p) 145
columnInitial (sbo.p) 146
columnLabel (dataextcols.p) 146
columnLabel (filter.p) 147
columnLabel (sbo.p) 147
columnLabelDefault (filter.p) 147
columnMandatory (sbo.p) 148
columnModified (dataextcols.p) 148
columnModified (sbo.p) 149
columnObjectHandle (sbo.p) 149
columnOperatorStyle (filter.p) 150
columnPrivateData (dataextcols.p) 150
columnPrivateData (sbo.p) 151
columnProperty (sbo.p) 151
columnProps (data.p) 152
columnQuerySelection (dataextcols.p) 152
columnQuerySelection (query.p) 153
columnQuerySelection (sbo.p) 154

columnReadOnly (dataextcols.p) 154
columnReadOnly (sbo.p) 155
columnStringValue (action.p) 155
columnStringValue (dataextcols.p) 156
columnStringValue (sbo.p) 156
columnStyleDefault (filter.p) 157
columnTable (dataextcols.p) 157
columnTable (query.p) 158
columnTable (sbo.p) 158
columnTooltip (filter.p) 159
columnValExp (dataextcols.p) 159
columnValExp (sbo.p) 159
columnValMsg (dataextcols.p) 160
columnValMsg (query.p) 160
columnValMsg (sbo.p) 160
columnValue (dataextcols.p) 161
columnValue (sbo.p) 161
columnWidth (dataextcols.p) 162
columnWidth (filter.p) 162
columnWidth (sbo.p) 163
columnWidthDefault (filter.p) 163
colValues (browser.p) 164
colValues (data.p) 164
colValues (query.p) 165
colValues (sbo.p) 165
Commit (data.i) 166
commitTransaction (data.p) 166
commitTransaction (sbo.p) 166
compareDBRow (query.p) 167
confirmCancel (containr.p) 167
confirmCancel (datavis.p) 168
confirmCommit (datavis.p) 168
confirmCommit (query.p) 168
confirmContinue (datavis.p) 169
confirmContinue (query.p) 169
confirmContinue (sbo.p) 170
confirmDelete (datavis.p) 170
confirmExit (containr.p) 171
confirmExit (datavis.p) 171
confirmOk (containr.p) 172
confirmOk (datavis.p) 172
confirmUndo (datavis.p) 173
confirmUndo (query.p) 173
constructMenu (toolbar.p) 173
constructMenuBand (toolbar.p) 174

constructObject (containr.p) 174
constructObjectMenus (toolbar.p) 175
constructToolbar (toolbar.p) 175
containedProperties (sbo.p) 176
copyColumns (data.p) 176
copyRecord (browser.p) 177
copyRecord (datavis.p) 178
copyRecord (viewer.p) 178
copyRow (data.p) 179
copyRow (sbo.p) 179
countButtons (panel.p) 180
create3DRule (toolbar.p) 180
createAttribute (xml.p) 180
createButton (toolbar.p) 181
createConsumers (consumer.p) 182
createControls (smart.p) 182
createDocument (router.p) 182
createDocument (xml.p) 183
createElement (xml.p) 183
createField (filter.p) 183
createLabel (combo.p) 185
createLabel (filter.p) 185
createLabel (lookup.p) 186
createLabel (select.p) 186
createMenuAction (toolbar.p) 187
createMenuBar (toolbar.p) 187
createMenuItem (toolbar.p) 188
createMenuTable (toolbar.p) 188
createMenuTable2 (toolbar.p) 189
createMenuToggle (toolbar.p) 190
createNode (xml.p) 191
createObjects (containr.p) 192
createOperator (filter.p) 193
createRow (data.p) 194
createRule (toolbar.p) 194
createSchemaAttributes (b2b.p) 194
createSchemaChildren (b2b.p) 195
createSchemaPath (b2b.p) 195
createSubMenu (toolbar.p) 196
createText (xml.p) 196
createToolBar (toolbar.p) 197
createToolbarAction (toolbar.p) 197
createToolbar (toolbar.p) 198
createToolbarBorder (toolbar.p) 198
currentMappedObject (sbo.p) 198

dataAvailable (browser.i) 199
dataAvailable (data.p) 199
dataAvailable (datavis.p) 200
dataAvailable (filter.p) 201
dataAvailable (query.p) 201
dataAvailable (sbo.p) 202
dataAvailable (select.p) 202
dataObjectHandle (sbo.p) 203
dataSource (b2b.p) 203
dataValue (filter.p) 203
dbColumnDataName (query.p) 204
dbColumnHandle (query.p) 204
defaultAction (browser.p) 204
defineAction (action.p) 205
defineDestination (consumer.p) 205
defineMapping (b2b.p) 206
deleteComplete (browser.p) 206
deleteComplete (sbo.p) 206
deleteDocument (xml.p) 207
deleteMenu (toolbar.p) 207
deleteNode (treeview.p) 207
deleteObjects (filter.p) 207
deletePage (containr.p) 208
deleteRecord (browser.p) 209
deleteRecord (datavis.p) 209
deleteRecordStatic (query.i) 210
deleteRow (data.p) 210
deleteRow (sbo.p) 210
deleteToolbar (toolbar.p) 211
deleteTree (treeview.p) 211
describeSchema (data.p) 211
destroyCombo (combo.p) 212
destroyLookup (lookup.p) 212
destroyObject (action.p) 212
destroyObject (appserver.p) 212
destroyObject (b2b.p) 213
destroyObject (browser.p) 213
destroyObject (combo.p) 213
destroyObject (consumer.p) 214
destroyObject (containr.p) 214
destroyObject (data.p) 214
destroyObject (design.p) 214
destroyObject (lookup.p) 215
destroyObject (messaging.p) 215
destroyObject (producer.p) 215

destroyObject (select.p) 215
destroyObject (smart.p) 216
destroyObject (toolbar.p) 216
destroyObject (xml.p) 217
destroySelection (select.p) 217
destroyServerObject (appserver.p) 217
destroyServerObject (data.p) 217
destroyServerObject (sbo.p) 218
disable_UI (combo.p) 218
disable_UI (lookup.p) 218
disableActions (panel.p) 218
disable_UI (select.p) 219
disableButton (lookup.p) 219
disableButton (select.p) 219
disableField (combo.p) 220
disableField (lookup.p) 220
disableField (select.p) 220
disableFields (browser.p) 221
disableFields (datavis.p) 221
disableFields (filter.p) 221
disableFields (viewer.p) 222
disableObject (treeview.p) 223
disableObject (visual.p) 223
disablePagesInFolder (containr.p) 223
disconnectObject (appserver.p) 224
displayActions (action.p) 224
displayCombo (combo.p) 224
displayFields (browser.p) 225
displayFields (viewer.p) 225
displayLinks (smart.p) 228
displayLookup (lookup.p) 228
displayObjects (datavis.i) 229
displayRecord (datavis.p) 229
doBuildUpd (data.i) 230
doCreateUpdate (data.i) 231
doEmptyModTable (data.i) 231
doEmptyTempTable (data.i) 232
doReturnUpd (data.i) 232
doUndoDelete (data.i) 233
doUndoRow (data.i) 233
doUndoTrans (data.i) 233
doUndoUpdate (data.i) 234
editInstanceProperties (smart.p) 234
emptyTree (treeview.p) 235
enableActions (panel.p) 235

enableButton (lookup.p) 236
enableButton (select.p) 236
enableField (combo.p) 236
enableField (lookup.p) 236
enableField (select.p) 237
enableFields (browser.p) 237
enableFields (datavis.p) 237
enableFields (filter.p) 238
enableFields (viewer.p) 238
enableObject (panel.p) 239
enableObject (treeview.p) 239
enableObject (visual.p) 239
enablePagesInFolder (containr.p) 240
endClientDataRequest (data.p) 240
endClientDataRequest (sbo.p) 240
endDocument (b2b.p) 241
endElement (b2b.p) 241
endMove (combo.p) 242
endMove (lookup.p) 242
endMove (select.p) 242
endTransactionValidate (datalogic.i) 242
enterCombo (combo.p) 243
enterLookup (lookup.p) 243
enterSelect (select.p) 243
errorHandler (consumer.p) 244
errorHandler (messaging.p) 244
errorMessage (action.p) 244
excludeColumns (query.p) 245
exitObject (smart.p) 246
fetchBatch (data.p) 247
fetchBatch (sbo.p) 247
fetchContainedData (sbo.p) 248
fetchContainedRows (sbo.p) 248
fetchDataSet (browser.p) 249
fetchDBRowForUpdate (query.p) 249
fetchDOProperties (sbo.p) 249
fetchFirst (data.p) 250
fetchFirst (query.p) 250
fetchFirst (sbo.p) 250
fetchLast (data.p) 251
fetchLast (query.p) 251
fetchLast (sbo.p) 251
fetchMessages (smart.p) 252
fetchNext (data.p) 252
fetchNext (query.p) 252

fetchNext (sbo.p) 253
fetchPrev (data.p) 253
fetchPrev (query.p) 253
fetchPrev (sbo.p) 253
fetchRow (data.p) 254
fetchRowIdent (data.p) 255
fieldLookup (filter.p) 255
fieldProperty (filter.p) 256
filterActive (browser.p) 256
filterContainerHandler (query.p) 257
filterState (toolbar.p) 257
findAction (action.p) 257
findCategory (action.p) 258
findDataRow (b2b.p) 258
findRow (data.p) 259
findRowObjUpd (data.p) 259
findRowWhere (data.p) 260
firstBufferName (query.p) 260
firstRowIds (data.p) 261
firstRowIds (query.p) 261
fixQueryString (smart.p) 262
formattedValue (select.p) 262
genContextList (data.p) 263
hasActiveAudit (browser.p) 263
hasActiveAudit (data.p) 263
hasActiveComments (browser.p) 264
hasActiveComments (data.p) 264
hasActiveGaTarget (panel.p) 264
hasForeignKeyChanged (data.p) 264
hasOneToOneTarget (data.p) 265
hideObject (combo.p) 265
hideObject (lookup.p) 265
hideObject (select.p) 265
hideObject (smart.p) 266
hideObject (toolbar.p) 267
hidePage (containr.p) 267
imageName (toolbar.p) 268
indexInformation (query.p) 268
initAction (action.p) 269
initDataObjectOrdering (sbo.i) 269
initializeBrowse (lookup.p) 269
initializeBrowse (select.p) 269
initializeCombo (combo.p) 270
initializeLogicObject (datalogic.i) 270
initializeLookup (lookup.p) 270

initializeObject (action.p) 270
initializeObject (b2b.p) 271
initializeObject (browser.p) 271
initializeObject (consumer.p) 272
initializeObject (containr.p) 272
initializeObject (data.p) 272
initializeObject (datavis.p) 273
initializeObject (field.p) 273
initializeObject (filter.p) 274
initializeObject (messaging.p) 274
initializeObject (panel.p) 275
initializeObject (producer.p) 275
initializeObject (query.p) 276
initializeObject (router.p) 276
initializeObject (sbo.p) 276
initializeObject (select.p) 277
initializeObject (smart.p) 277
initializeObject (toolbar.p) 278
initializeObject (tvcontroller.p) 278
initializeObject (viewer.p) 278
initializeObject (visual.p) 279
initializeSelection (select.p) 279
initializeServerObject (appserver.p) 280
initializeServerObject (data.p) 280
initializeServerObject (sbo.p) 280
initializeVisualContainer (containr.p) 280
initPages (containr.p) 281
initProps (query.i) 282
insertExpression (combo.p) 283
insertExpression (lookup.p) 284
insertExpression (query.p) 284
insertFieldProperty (filter.p) 285
insertMenu (toolbar.p) 286
instancePropertyList (smart.p) 286
internalSchemaFile (router.p) 287
isNodeExpanded (treeview.p) 288
isObjQuoted (smart.p) 288
isUpdateActive (containr.p) 289
isUpdateActive (data.p) 289
isUpdateActive (datavis.p) 290
isUpdatePending (data.p) 290
isUpdatePending (sbo.p) 291
launchFolderWindow (browser.p) 292
leaveCombo (combo.p) 292
leaveLookup (lookup.p) 292

leaveSelect (select.p) 293
linkHandles (design.p) 293
linkHandles (smart.p) 294
linkProperty (smart.p) 296
linkState (panel.p) 297
linkState (toolbar.p) 297
loadActions (action.p) 297
loadBands (action.p) 298
loadCategories (action.p) 298
loadImage (treeview.p) 299
loadObjectBands (action.p) 299
loadPanel (panel.p) 300
loadProducerSchema (b2b.p) 300
loadSchema (b2b.p) 300
loadToolbar (toolbar.p) 300
loadToolbarBands (action.p) 301
mapNode (b2b.p) 302
mappedEntry (smart.p) 303
menuHandle (toolbar.p) 303
messageHandler (consumer.p) 304
messageNumber (smart.p) 305
modifyDisabledActions (toolbar.p) 306
modifyListProperty (smart.p) 306
modifyUserLinks (smart.p) 308
movebuttons (toolbar.p) 309
moveMenu (toolbar.p) 309
moveMenuChildren (toolbar.p) 310
newDataObjectRow (sbo.p) 310
newQueryString (combo.p) 311
newQueryString (lookup.p) 312
newQueryString (query.p) 313
newQueryValidate (query.p) 314
newQueryWhere (query.p) 315
newRowObject (data.p) 315
newWhereClause (combo.p) 316
newWhereClause (lookup.p) 317
newWhereClause (query.p) 317
nodeHandle (xml.p) 318
nodeType (xml.p) 318
normalizeActionData (action.p) 318
NotFoundError (b2b.p) 319
notifyPage (containr.p) 319
numParameters (b2b.p) 320
obtainInMsgTarget (router.p) 320
.offEnd (browser.p) 321

offHome (browser.p) . . . 321
okObject (containr.p) . . . 321
okObject (datavis.p) . . . 322
okToContinueProcedure (datavis.p) . . . 322
onChoose (panel.p) . . . 323
onChoose (toolbar.p) . . . 323
oneObjectLinks (smart.p) . . . 323
onMenuDrop (toolbar.p) . . . 324
onValueChanged (toolbar.p) . . . 324
openDataQuery (data.p) . . . 324
openQuery (data.p) . . . 325
openQuery (query.p) . . . 325
openQuery (sbo.p) . . . 325
ownerElement (xml.p) . . . 325
pageNTargets (containr.p) . . . 326
parentNode (xml.p) . . . 326
passThrough (containr.p) . . . 327
populateTree (treeview.p) . . . 327
postCreateObjects (sbo.p) . . . 329
postTransactionValidate (datalogic.i) . . . 329
prepareErrorsForReturn (data.i) . . . 329
prepareErrorsForReturn (sbo.p) . . . 330
prepareQueriesForFetch (sbo.p) . . . 330
prepareQuery (data.p) . . . 331
prepareQuery (query.p) . . . 331
preTransactionValidate (datalogic.i) . . . 332
printToCrystal (data.p) . . . 332
processAction (visual.p) . . . 333
processCDataSection (xml.p) . . . 333
processComment (xml.p) . . . 333
processDestinations (consumer.p) . . . 334
processDocument (xml.p) . . . 334
processElement (xml.p) . . . 335
processFileRefs (router.p) . . . 335
processMappings (b2b.p) . . . 335
processMessages (b2b.p) . . . 336
processRoot (xml.p) . . . 336
processText (xml.p) . . . 336
produceAttributes (b2b.p) . . . 337
produceChildren (b2b.p) . . . 337
produceDocument (b2b.p) . . . 337
propertyType (smart.p) . . . 338
pushTableAndValidate (data.i) . . . 338
putParam (tvcontroller.p) . . . 339
queryOpened (select.p) . . . 339

queryPosition (datavis.p) 340
queryPosition (panel.p) 340
queryPosition (sbo.p) 341
queryPosition (toolbar.p) 341
receiveHandler (b2b.p) 341
receiveHandler (xml.p) 342
receiveReplyHandler (xml.p) 342
refetchDBRow (query.p) 343
refreshBrowse (browser.p) 343
refreshBrowse (sbo.p) 343
refreshChildDependancies (combo.p) 344
refreshCombo (combo.p) 344
registerLinkedObjects (sbo.p) 344
refreshObject (select.p) 345
refreshRow (data.p) 345
refreshRowident (query.p) 345
registerObject (sbo.p) 346
releaseDBRow (query.p) 346
ReleaseDC (treeview.i) 347
removeAllLinks (smart.p) 347
removeForeignKey (query.p) 347
removeLink (smart.p) 348
removePageNTarget (containr.p) 349
removeQuerySelection (query.p) 349
removeQuerySelection (sbo.p) 350
removeSpace (filter.p) 351
replyHandler (producer.p) 351
repositionDataSource (select.p) 351
repositionObject (smart.p) 352
repositionObject (toolbar.p) 352
repositionObject (treeview.p) 352
repositionRowObject (data.p) 353
repositionRowObject (sbo.p) 353
resetCommit (panel.p) 353
resetFields (filter.p) 354
resetNavigation (panel.p) 354
resetQuery (sbo.p) 354
resetRecord (browser.p) 355
resetRecord (datavis.p) 355
resetTableio (panel.p) 356
resetTargetActions (panel.p) 356
resetTargetActions (toolbar.p) 356
resetToolbar (toolbar.p) 357
resizeBrowse (browser.p) 357
resizeObject (browser.p) 357

resizeObject (combo.p) 358
resizeObject (field.p) 359
resizeObject (lookup.p) 359
resizeObject (panel.p) 360
resizeObject (select.p) 360
resizeObject (toolbar.p) 361
resizeObject (treeview.p) 361
resizeWindow (containr.p) 362
resolveBuffer (query.p) 362
restartServerObject (appserver.p) 363
restartServerObject (data.p) 363
restartServerObject (sbo.p) 364
retrieveFilter (data.p) 364
returnFocus (smart.p) 364
returnParentFieldValues (combo.p) 365
returnParentFieldValues (lookup.p) 365
reviewMessages (smart.p) 365
routeBytesMessage (router.p) 366
routeDocument (router.p) 366
routeMessage (router.p) 367
rowAvailable (data.p) 367
rowDisplay (browser.p) 368
rowidWhere (query.p) 368
rowidWhereCols (query.p) 369
rowNotFoundError (b2b.p) 369
rowObjectState (data.p) 370
rowObjectState (toolbar.p) 370
rowObjectValidate (datalogic.i) 371
rowSelected (lookup.p) 371
rowSelected (select.p) 371
rowValues (data.p) 372
rowVisible (browser.p) 372
runInfo (toolbar.p) 373
runServerObject (appserver.p) 373
runServerProcedure (appserver.p) 374
saveContextAndDestroy (data.p) 375
schemaField (b2b.p) 375
searchTrigger (browser.p) 376
selectFirstNode (treeview.p) 376
selectNode (treeview.p) 376
selectPage (containr.p) 377
sendHandler (b2b.p) 378
sendHandler (xml.p) 378
sendMessage (b2b.p) 379
sendMessage (msghandler.p) 379

sendMessage (producer.p) 379
SendMessageA (treeview.i) 380
sensitizeActions (panel.p) 381
sendReplyHandler (xml.p) 381
sendRows (data.p) 382
sensitizeActions (toolbar.p) 383
serverCommit (data.i) 383
serverCommit (datamessage.p) 384
serverCommit (logic.i) 384
serverCommitTransaction (datamessage.p) 385
serverCommitTransaction (sbo.i) 385
serverContainedSendRows (sbo.p) 386
serverFetchContainedData (datamessage.p) 387
serverFetchContainedRows (datamessage.p) 388
serverFetchContainedData (sbo.p) 389
serverFetchContainedRows (sbo.p) 389
serverFetchDOProperties (sbo.p) 390
serverFetchRowObjUpdTable (data.p) 390
serverSendRows (data.i) 391
serverSendRows (datamessage.p) 392
ShellExecuteA (treeview.i) 393
showDataMessages (datavis.p) 394
showDataMessages (filter.p) 394
showDataMessagesProcedure (datavis.p) 395
showMessage (smart.p) 395
showMessageProcedure (smart.p) 396
showQuery (combo.p) 396
showTVCError (tvcontroller.p) 397
showTVError (treeview.p) 397
Signature (smart.p) 397
startSearch (browser.p) 398
startServerObject (appserver.p) 398
startServerObject (data.p) 399
startServerObject (sbo.p) 399
start-super-proc (smart.i) 399
startB2BObject (router.p) 399
startDataRow (b2b.p) 400
startElement (b2b.p) 400
startFilter (query.p) 401
startServerObject (appserver.p) 401
startServerObject (data.p) 401
startServerObject (sbo.p) 401
startWaitFor (consumer.p) 402
stopHandler (consumer.p) 402
storeNodeValue (b2b.p) 403

storeParameterNode (b2b.p) 403
storeParameterValue (b2b.p) 404
storePendingSensitivity (toolbar.p) 405
stripCalcs (browser.p) 405
submitCommit (data.p) 406
submitForeignKey (data.p) 406
submitRow (data.p) 408
submitRow (sbo.p) 408
submitValidation (data.p) 409
substituteActionText (toolbar.p) 409
supportedObjects (toolbar.p) 410
synchronizeProperties (data.p) 410
tableOut (data.p) 411
targetActions (panel.p) 412
targetActions (toolbar.p) 412
targetPage (containr.p) 412
targetProcedure (b2b.p) 413
toggleData (smart.p) 413
toolbar (browser.p) 413
toolbar (containr.p) 414
toolbar (viewer.p) 415
transferDBRow (query.p) 415
transferToExcel (data.p) 416
translateBrowseColumns (lookup.p) 416
unbindServer (appserver.p) 417
unBlankFillin (filter.p) 418
unBlankLogical (filter.p) 418
undoClientUpdate (data.i) 419
undoTransaction (data.p) 419
undoTransaction (sbo.p) 419
updateActive (containr.p) 420
updateActive (toolbar.p) 420
updateAddQueryWhere (data.p) 421
updateCategoryLists (toolbar.p) 421
updateManualForeignFields (data.p) 422
updateMode (datavis.p) 422
updateQueryPosition (data.p) 423
updateRecord (browser.p) 423
updateRecord (datavis.p) 424
updateRow (data.p) 424
updateState (browser.p) 425
updateState (data.p) 426
updateState (datavis.p) 426
updateState (panel.p) 427
updateState (sbo.p) 427

updateState (toolbar.p) 428
updateState (tvcontroller.p) 428
updateState (viewer.p) 429
updateStates (toolbar.p) 429
updateTitle (browser.p) 429
updateTitle (datavis.p) 430
updateTree (treeview.p) 430
validateBuffer (action.p) 431
validateFields (datavis.p) 431
valueChanged (combo.p) 432
valueChanged (lookup.p) 432
valueChanged (select.p) 432
valueChanged (viewer.p) 433
viewHideActions (panel.p) 433
viewHideActions (toolbar.p) 434
viewObject (browser.p) 434
viewObject (combo.p) 435
viewObject (containr.p) 435
viewObject (lookup.p) 436
viewObject (select.p) 436
viewObject (smart.p) 436
viewObject (toolbar.p) 437
viewPage (containr.p) 437
viewRecord (viewer.p) 438
whereClauseBuffer (combo.p) 439
whereClauseBuffer (lookup.p) 439
whereClauseBuffer (query.p) 440
windowDropDownList (toolbar.p) 440
windowEnabled (toolbar.p) 441
windowFocus (toolbar.p) 441
windowListMenu (toolbar.p) 442

SmartObject API Reference For WebSpeed 443
Paths To Source Files 444

Alphabetical Listing Of WebSpeed-specific API Routines. 445
addColumnLink (wbtable.p) 446
addContextFields (webrep.p) 449
addSearchCriteria (webrep.p) 449
addTDModifier (wbtable.p) 450
anyMessage (webrep.p) 450
assignColumnFormat (webrep.p) 451
assignColumnHelp (webrep.p) 451
assignColumnLabel (webrep.p) 452
assignColumnWidth (webrep.p) 452

assignExtentAttribute (webrep.p) 453
assignFields (html-map.p) 454
assignFields (wbdata.p) 455
assignTDModifier (wbtable.p) 456
bufferHandle (html-map.p) 456
bufferHandle (webrep.p) 457
columnDataType (webrep.p) 457
columnFormat (webrep.p) 457
columnHandle (webrep.p) 458
columnHelp (webrep.p) 458
columnHTMLName (html-map.p) 458
columnHTMLName (webrep.p) 459
columnLabel (webrep.p) 459
columnReadOnly (webrep.p) 460
columnStringValue (webrep.p) 460
columnTable (html-map.p) 460
columnTDModifier (wbtable.p) 461
columnValMsg (wbdata.p) 461
constructObject (html-map.p) 462
dataAvailable (html-map.p) 462
deleteBuffer (wbdata.p) 463
deleteOffsets (html-map.p) 463
deleteRow (wbdata.p) 464
destroy (admweb.p) 464
destroyDataObject (webrep.p) 464
destroyObject (admweb.p) 465
destroyObject (webrep.p) 465
disconnectObject (webrep.p) 465
dispatchUtilityProc (html-map.p) 466
displayFields (html-map.p) 467
enableFields (html-map.p) 468
exclusiveLockBuffer (wbdata.p) 468
extentAttribute (webrep.p) 469
extentValue (webrep.p) 469
fetchCurrent (webrep.p) 470
fetchFirst (webrep.p) 470
fetchLast (wbtable.p) 470
fetchLast (webrep.p) 471
fetchNext (wbtable.p) 471
fetchNext (webrep.p) 471
fetchPrev (wbtable.p) 472
fetchPrev (webrep.p) 472
fieldExpression (webrep.p) 472
findRecords (html-map.p) 473
htmAssociate (html-map.p) 474

HTMLAlert (wbrep.p) 474
HTMLColumn (wbtable.p) 475
HTMLSetFocus (wbrep.p) 475
HTMLTable (wbtable.p) 476
initializeObject (html-map.p) 476
inputFields (html-map.p) 476
joinExternalTables (wbrep.p) 477
joinForeignFields (wbrep.p) 478
lockRow (wbdata.p) 478
openQuery (webrep.p) 479
outputFields (html-map.p) 479
pageBackward (wbtable.p) 480
processWebRequest (wbdata.p) 480
processWebRequest (wbtable.p) 481
processWebRequest (webrep.p) 482
readOffsets (html-map.p) 483
removeEntry (webrep.p) 483
reOpenQuery (webrep.p) 483
rowidExpression (webrep.p) 484
showDataMessages (webrep.p) 484
startDataObject (webrep.p) 485
timingOut (admweb.p) 485
urlJoinParams (webrep.p) 486
urlLink (webrep.p) 486
validateColumns (wbdata.p) 487
validateColumnValue (webrep.p) 488

Properties Glossary Glossary–1

Index Index–1

Figures

Figure 1: ADM2 Class Tree Diagram Showing Inheritance Relationships 4
Figure 2: Commented Source Code Of GET Pseudo-function 32

Tables

Table 1: SmartObjects and Their Files 5
Table 2: Public Properties By Name: A 33
Table 3: Public Properties By Name: B 35
Table 4: Public Properties By Name: C 36
Table 5: Public Properties By Name: D 38
Table 6: Public Properties By Name: E 42
Table 7: Public Properties By Name: F 43
Table 8: Public Properties By Name: G 45
Table 9: Public Properties By Name: H 45
Table 10: Public Properties By Name: I 45
Table 11: Public Properties By Name: J 46
Table 12: Public Properties By Name: K 46
Table 13: Public Properties By Name: L 47
Table 14: Public Properties By Name: M 47
Table 15: Public Properties By Name: N 48
Table 16: Public Properties By Name: O 50
Table 17: Public Properties By Name: P 51
Table 18: Public Properties By Name: Q 52
Table 19: Public Properties By Name: R 53
Table 20: Public Properties By Name: S 54
Table 21: Public Properties By Name: T 56
Table 22: Public Properties By Name: U 57
Table 23: Public Properties By Name: V 58
Table 24: Public Properties By Name: W 58

Preface

Purpose

This manual is an API reference manual for the Progress Application Development Model (ADM2). It describes the procedures and functions that define the default behavior of Progress SmartObjects.

Audience

This book is intended for Progress ADM2 application developers. It presumes a reasonably thorough knowledge of the underlying 4GL language and Progress ProVision™.

Organization Of This Manual

ADM2 SmartObject API Reference

Provides a brief discussion of the ADM architecture, illustration of the class hierarchy tree, each of the current SmartObjects and the files that define them, the two syntaxes for property reads and writes, and tables showing properties defined in each of the class files.

Alphabetical Listing of API Routines

Offers a brief description, including calling sequence, of each of the routines in the various super-procedure files. The listing includes the get/set property routines as a convenience, although their names can virtually always be predicted from the name of the property on which they operate.

SmartObject API Reference For WebSpeed

Provides a brief introduction to the SmartObjects for WebSpeed and the files that define them.

Alphabetical Listing Of WebSpeed-specific API Routines

Gives a brief description, including calling sequence, of each of the routines in the various WebSpeed super-procedure files.

"Properties Glossary"

Provides a listing of the ADM2 properties that can be accessed with the {get} and {set} pseudo-functions.

ADM2 classes are more similar to procedure-oriented libraries—C-language run-time libraries, for example—than to classes in object-oriented theory. In general, ADM2 classes do not hide their property storage or their methods from one another. For this reason, as well as for convenience in lookup, the routines are listed in alphabetic order without regard to class or source. For a list of routines by class, see the index.

How To Use This Manual

The API entries in this reference manual are laid out in the following format. Not all entries have notes and examples:

call_name (file_location)

Description of the call's purpose.

Parameters:

param_type param_name AS *data_type*

Parameter description.

Returns: *return_value*

Notes:

- Notes on use of the call.

Examples:

A code example illustrating the use of the call.

Typographical Conventions

This manual uses the following typographical conventions:

- **Bold typeface** indicates:
  - Commands or characters that the user types
  - That a word carries particular weight or emphasis
- *Italic typeface* indicates:
  - Progress variable information that the user supplies
  - New terms
  - Titles of complete publications
- `Monospaced typeface` indicates:
  - Code examples
  - System output
  - Operating system filenames and pathnames

The following typographical conventions are used to represent keystrokes:

- Small capitals are used for Progress key functions and generic keyboard keys.

  END-ERROR, GET, GO
  ALT, CTRL, SPACEBAR, TAB

- When you have to press a combination of keys, they are joined by a dash. You press and hold down the first key, then press the second key.

  CTRL-X

- When you have to press and release one key, then press another key, the key names are separated with a space.

  ESCAPE H
  ESCAPE CURSOR-LEFT

Syntax Notation

The syntax for each component follows a set of conventions:

- Uppercase words are keywords. Although they are always shown in uppercase, you can use either uppercase or lowercase when using them in a procedure.

  In this example, ACCUM is a keyword:

  SYNTAX

  ```
  ACCUM aggregate expression
  ```

- Italics identify options or arguments that you must supply. These options can be defined as part of the syntax or in a separate syntax identified by the name in italics. In the ACCUM function above, the *aggregate* and *expression* options are defined with the syntax for the ACCUM function in the *Progress Language Reference*.

- You must end all statements (except for DO, FOR, FUNCTION, PROCEDURE, and REPEAT) with a period. DO, FOR, FUNCTION, PROCEDURE, and REPEAT statements can end with either a period or a colon, as in this example:

  ```
  FOR EACH Customer:
     DISPLAY Name.
  END.
  ```

- Square brackets (**[ ]**) around an item indicate that the item, or a choice of one of the enclosed items, is optional.

  In this example, STREAM *stream*, UNLESS-HIDDEN, and NO-ERROR are optional:

  SYNTAX

  ```
  DISPLAY [ STREAM stream ] [ UNLESS-HIDDEN ] [ NO-ERROR ]
  ```

  In some instances, square brackets are not a syntax notation, but part of the language.

For example, this syntax for the INITIAL option uses brackets to bound an initial value list for an array variable definition. In these cases, normal text brackets ([ ]) are used:

SYNTAX

```
INITIAL [ constant [ , constant ] ... ]
```

NOTE: The ellipsis (**...**) indicates repetition, as shown in a following description.

- Braces (**{ }**) around an item indicate that the item, or a choice of one of the enclosed items, is required.

  In this example, you must specify the items BY and *expression* and can optionally specify the item DESCENDING, in that order:

  SYNTAX

  ```
  { BY expression [ DESCENDING ] }
  ```

  In some cases, braces are not a syntax notation, but part of the language.

  For example, a called external procedure must use braces when referencing arguments passed by a calling procedure. In these cases, normal text braces ({ }) are used:

  SYNTAX

  ```
  { &argument-name }
  ```

- A vertical bar (**|**) indicates a choice.

  In this example, EACH, FIRST, and LAST are optional, but you can only choose one:

  SYNTAX

  ```
  PRESELECT [ EACH | FIRST | LAST ] record-phrase
  ```

In this example, you must select one of *logical-name* or *alias*:

SYNTAX

```
CONNECTED ( { logical-name | alias } )
```

- Ellipses (**. . .**) indicate that you can choose one or more of the preceding items. If a group of items is enclosed in braces and followed by ellipses, you must choose one or more of those items. If a group of items is enclosed in brackets and followed by ellipses, you can optionally choose one or more of those items.

  In this example, you must include two expressions, but you can optionally include more. Note that each subsequent expression must be preceded by a comma:

  SYNTAX

  ```
  MAXIMUM ( expression , expression [ , expression ] ... )
  ```

  In this example, you must specify MESSAGE, then at least one of *expression* or SKIP, but any additional number of *expression* or SKIP is allowed:

  SYNTAX

  ```
  MESSAGE { expression | SKIP [ (n) ] } ...
  ```

  In this example, you must specify {*include-file*, then optionally any number of *argument* or &*argument-name* = "*argument-value*", and then terminate with }:

  SYNTAX

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

- In some examples, the syntax is too long to place in one horizontal row. In such cases, **optional** items appear individually bracketed in multiple rows in order, left-to-right and top-to-bottom. This order generally applies, unless otherwise specified. **Required** items also appear on multiple rows in the required order, left-to-right and top-to-bottom. In cases where grouping and order might otherwise be ambiguous, braced (required) or bracketed (optional) groups clarify the groupings.

In this example, WITH is followed by several optional items:

SYNTAX

```
WITH [ ACCUM max-length ] [ expression DOWN ]
  [ CENTERED ] [ n COLUMNS ] [ SIDE-LABELS ]
  [ STREAM-IO ]
```

In this example, ASSIGN requires one of two choices: either one or more of *field*, or one of *record*. Other options available with either *field* or *record* are grouped with braces and brackets. The open and close braces indicate the required order of options:

SYNTAX

```
ASSIGN {   { [ FRAME frame ]
               { field [ = expression ] }
               [ WHEN expression ]
           } ...
       |  { record [ EXCEPT field ... ] }
     }
```

Example Procedures

This manual provides numerous example procedures that illustrate syntax and concepts. Examples use the following conventions:

- They appear in boxes with borders.
- If they are available online, the name of the procedure appears above the left corner of the box and starts with a prefix associated with the manual that references it, as follows:
  - `e-` — *Progress External Program Interfaces*, for example, `e-ddeex1.p`
  - `lt-` — *Progress Language Tutorials*, for example, `lt-05-s3.p`
  - `p-` — *Progress Programming Handbook*, for example, `p-br01.p`
  - `r-` — *Progress Language Reference*, for example, `r-dynbut.p`

  If the name does not start with a listed prefix, the procedure is not available online.
- If they are not available online, they compile as shown, but might not execute for lack of completeness.

Accessing Files In Procedure Libraries

Documentation examples are stored in procedure libraries, `prodoc.pl` and `prohelp.pl`, in the `src` directory where Progress is installed.

You **must** first create all subdirectories required by a library before attempting to extract files from the library. You can see what directories and subdirectories a library needs by using the PROLIB -list command to view the contents of the library. See the *Progress Client Deployment Guide* for more details on the PROLIB utility.

Creating a Listing Of the Procedure Libraries

Creating a listing of the source files from a procedure library involves running PROENV to set up your Progress environment, and running PROLIB.

1 ♦ From the Control Panel or the Progress Program Group, double-click the Proenv icon.

2 ♦ The Proenv Window appears, with the proenv prompt.

Running Proenv sets the DLC environment variable to the directory where you installed Progress (by default, `C:\Program Files\Progress`). Proenv also adds the DLC environment variable to your PATH environment variable and adds the bin directory (`PATH=%DLC%;%DLC%\bin;%PATH%`).

3 ♦ Enter the following command at the proenv prompt to create the text file `prodoc.txt` which contains the file listing for the prodoc.pl library.

```
PROLIB %DLC%\src\prodoc.pl -list > prodoc.txt
```

Extracting Source Files From Procedure Libraries On Windows Platforms

Extracting source files from a procedure library involves running PROENV to set up your Progress environment, creating the directory structure for the files you want to extract, and running PROLIB.

1 ♦ From the Control Panel or the Progress Program Group, double-click the Proenv icon.

2 ♦ The Proenv Window appears, with the proenv prompt.

Running Proenv sets the DLC environment variable to the directory where you installed Progress (by default, `C:\Program Files\Progress`). Proenv also adds the DLC environment variable to your PATH environment variable and adds the bin directory (`PATH=%DLC%;%DLC%\bin;%PATH%`).

3 ♦ Enter the following command at the proenv prompt to create the `prodoc` directory in your Progress working directory (by default, `C:\Progress\Wrk`):

```
MKDIR prodoc
```

4 ♦ Create the `langref` directory under `prodoc`:

```
MKDIR prodoc\langref
```

5 ♦ To extract all examples in a procedure library directory, run the PROLIB utility. Note that you must use double quotes because “Program Files” contains an embedded space:

```
PROLIB "%DLC%\src\prodoc.pl" -extract prodoc\langref\*.*
```

PROLIB extracts all examples into `prodoc\langref`.

To extract one example, run PROLIB and specify the file that you want to extract as it is stored in the procedure library:

```
PROLIB "%DLC%\src\prodoc.pl" -extract prodoc/langref/r-syshlp.p
```

PROLIB extracts `r-syshlp.p` into `prodoc\langref`.

Extracting Source Files From Procedure Libraries On UNIX Platforms

To extract `p-wrk1.p` from `prodoc.pl`, a procedure library, follow these steps at the UNIX system prompt:

1 ♦ Run the PROENV utility:

```
install-dir/dlc/bin/proenv
```

Running proenv sets the DLC environment variable to the directory where you installed Progress (by default, `/usr/dlc`). The proenv utility also adds the bin directory under the DLC environment variable to your PATH environment variable (`PATH=$DLC/bin:$PATH`).

2 ♦ At the proenv prompt, create the `prodoc` directory in your Progress working directory:

```
mkdir prodoc
```

3 ♦ Create the `proghand` directory under `prodoc`:

```
mkdir prodoc/proghand
```

4 ♦ To extract all examples in a procedure library directory, run the PROLIB utility:

```
prolib $DLC/src/prodoc.pl -extract prodoc/proghand/*.*
```

PROLIB extracts all examples into `prodoc/proghand`.

To extract one example, run PROLIB and specify the file that you want to extract as it is stored in the procedure library:

```
prolib $DLC/src/prodoc.pl -extract prodoc/proghand/p-wrk-1.p
```

PROLIB extracts `p-wrk-1.p` into `prodoc/proghand`.

Progress Messages

Progress displays several types of messages to inform you of routine and unusual occurrences:

- Execution messages inform you of errors encountered while Progress is running a procedure (for example, if Progress cannot find a record with a specified index field value).
- Compile messages inform you of errors found while Progress is reading and analyzing a procedure prior to running it (for example, if a procedure references a table name that is not defined in the database).
- Startup messages inform you of unusual conditions detected while Progress is getting ready to execute (for example, if you entered an invalid startup parameter).

After displaying a message, Progress proceeds in one of several ways:

- Continues execution, subject to the error-processing actions that you specify, or that are assumed, as part of the procedure. This is the most common action taken following execution messages.
- Returns to the Progress Procedure Editor so that you can correct an error in a procedure. This is the usual action taken following compiler messages.
- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.
- Terminates the current session.

Progress messages end with a message number in parentheses. In this example, the message number is 200:

```
** Unknown table name table. (200)
```

On the Windows platform, use Progress online help to get more information about Progress messages. Many Progress tools include the following Help menu options to provide information about messages:

- Choose Help→Recent Messages to display detailed descriptions of the most recent Progress message and all other messages returned in the current session.
- Choose Help→Messages, then enter the message number to display a description of any Progress message. (If you encounter an error that terminates Progress, make a note of the message number before restarting.)
- In the Procedure Editor, press the **HELP** key (**F2** or **CTRL-W**).

On the UNIX platfrom, use the Progress PRO command to start a single-user mode character Progress client session and view a brief description of a message by providing its number. Follow these steps:

1 ♦ Start the Progress Procedure Editor:

```
install-dir/dlc/bin/pro
```

2 ♦ Press **F3** to access the menu bar, then choose Help→Messages.

3 ♦ Type the message number, and press **ENTER**. Details about that message number appear.

4 ♦ Press **F4** to close the message, press **F3** to access the Procedure Editor menu, and choose File→Exit.

Other Useful Documentation

This section lists Progress Software Corporation documentation that you might find useful. Unless otherwise specified, these manuals support both Windows and Character platforms and are provided in electronic documentation format on CD-ROM.

Getting Started

Progress Electronic Documentation Installation and Configuration Guide (Hard copy only)

A booklet that describes how to install the Progress EDOC viewer and collection on UNIX and Windows.

Progress Installation and Configuration Guide Version 9 for UNIX

A manual that describes how to install and set up Progress Version 9.1 for the UNIX operating system.

Progress Installation and Configuration Guide Version 9 for Windows

A manual that describes how to install and set up Progress Version 9.1 for all supported Windows and Citrix MetaFrame operating systems.

Progress Version 9 Product Update Bulletin

A bulletin that provides a list of new and changed features by release, a list and description of changes to documentation by release, and critical information about product changes that might require changes to existing code and configurations.

This bulletin also provides information about where to go for detailed information about the new and changed features and documentation.

Progress Application Development Environment — Getting Started (Windows only)

A practical guide to graphical application development within the Progress Application Development Environment (ADE). This guide includes an overview of the ADE and its tools, an overview of Progress SmartObject technology, and tutorials and exercises that help you better understand SmartObject technology and how to use the ADE to develop applications.

Progress Language Tutorial for Windows and *Progress Language Tutorial for Character*

Platform-specific tutorials designed for new Progress users. The tutorials use a step-by-step approach to explore the Progress application development environment using the 4GL.

Progress Master Glossary for Windows and *Progress Master Glossary for Character* (EDOC only)

Platform-specific master glossaries for the Progress documentation set. These books are in electronic format only.

Progress Master Index and Glossary for Windows and *Progress Master Index and Glossary for Character* (Hard copy only)

Platform-specific master indexes and glossaries for the Progress hard-copy documentation set.

Progress Startup Command and Parameter Reference

A reference manual that describes the Progress startup and shutdown commands that you use at the command line, and the startup parameters that you use for Progress processes. This guide also provides information about parameter usage and parameter files.

Welcome to Progress (Hard copy only)

A booklet that explains how Progress software and media are packaged. An icon-based map groups the documentation by functionality, providing an overall view of the documentation set. *Welcome to Progress* also provides descriptions of the various services Progress Software Corporation offers.

Development Tools

Progress ADM 2 Guide

A guide to using the Application Development Model, Version 2 (ADM 2) application architecture to develop Progress applications. It includes instructions for building and using Progress SmartObjects.

Progress AppBuilder Developer's Guide (Windows only)

A programmer's guide to using the Progress AppBuilder visual layout editor. AppBuilder is a Rapid Application Development (RAD) tool that can significantly reduce the time and effort required to create Progress applications.

Progress Basic Database Tools (Character only; information for Windows is in online help)

A guide for the Progress Database Administration tools, such as the Data Dictionary.

Progress Basic Development Tools (Character only; information for Windows is in online help)

A guide for the Progress development toolset, including the Progress Procedure Editor and the Application Compiler.

Progress Debugger Guide

A guide for the Progress Application Debugger. The Debugger helps you trace and correct programming errors by allowing you to monitor and modify procedure execution as it happens.

Progress Help Development Guide (Windows only)

A guide that describes how to develop and integrate an online help system for a Progress application.

Progress Translation Manager Guide (Windows only)

A guide that describes how to use the Progress Translation Manager tool to manage the entire process of translating the text phrases in Progress applications.

Progress Visual Translator Guide (Windows only)

A guide that describes how to use the Progress Visual Translator tool to translate text phrases from procedures into one or more spoken languages.

Reporting Tools

Progress Report Builder Deployment Guide (Windows only)

An administration and development guide for generating Report Builder reports using the Progress Report Engine.

Progress Report Builder Tutorial (Windows only)

A tutorial that provides step-by-step instructions for creating eight sample Report Builder reports.

Progress Report Builder User's Guide (Windows only)

A guide for generating reports with the Progress Report Builder.

Progress Results Administration and Development Guide (Windows only)

A guide for system administrators that describes how to set up and maintain the Results product in a graphical environment. This guide also describes how to program, customize, and package Results with your own products. In addition, it describes how to convert character-based Results applications to graphical Results applications.

Progress Results User's Guide for Windows and *Progress Results User's Guide for Unix*

Platform-specific guides for users with little or no programming experience that explain how to query, report, and update information with Results. Each guide also helps advanced users and application developers customize and integrate Results into their own applications.

4GL

Building Distributed Applications Using the Progress AppServer

A guide that provides comprehensive information about building and implementing distributed applications using the Progress AppServer. Topics include basic product information and terminology, design options and issues, setup and maintenance considerations, 4GL programming details, and remote debugging.

Progress External Program Interfaces

A guide to the external programming interfaces supported by Progress. This manual covers the Host Language Call (HLC) Interface, the system clipboard, named pipes, shared libraries and DLLS, Windows Dynamic Data Exchange (DDE), COM objects, ActiveX Automation, ActiveX controls, sockets, XML, SAX, and the SonicMQ 4GL Adapter.

Progress Internationalization Guide

A guide to developing Progress applications for markets worldwide. The guide covers both internationalization—writing an application so that it adapts readily to different locales (languages, cultures, or regions)—and localization—adapting an application to different locales.

Progress Language Reference

A three-volume reference set that contains extensive descriptions and examples for each statement, phrase, function, operator, widget, attribute, method, and event in the Progress language.

Progress on the Web

A manual that describes how to use the new WebClient, AppServer Internet Adapter, SmartObjects, and SonicMQ Adapter to create applications tailored for Internet, intranet, and extranet environments.

Progress Programming Handbook

A two-volume handbook that details advanced Progress programming techniques.

Database

Progress Database Design Guide

A guide that uses a sample database and the Progress Data Dictionary to illustrate the fundamental principles of relational database design. Topics include relationships, normalization, indexing, and database triggers.

Progress Database Administration Guide and Reference

This guide describes Progress database administration concepts and procedures. The procedures allow you to create and maintain your Progress databases and manage their performance.

DataServers

Progress DataServer Guides

These guides describe how to use the DataServers to access non-Progress databases. They provide instructions for building the DataServer modules, a discussion of programming considerations, and a tutorial.

Each DataServer has its own guide as follows:

- *Progress/400 Product Guide*
- *Progress DataServer for Microsoft SQL Server Guide*
- *Progress DataServer for ODBC Guide*
- *Progress DataServer for ORACLE Guide*

MERANT ODBC Branded Driver Reference

The Enterprise DataServer for ODBC includes MERANT ODBC drivers for all the supported data sources. For configuration information, see the MERANT documentation, which is available as a PDF file in *installation-path*\odbc. To read this file you must have the Adobe Acrobat Reader Version installed on your system. If you do not have the Adobe Acrobat Reader, you can download it from the Adobe Web site at: http://www.adobe.com/products/acrobat/readstep.html.

SQL-89/Open Access

Progress Embedded SQL-89 Guide and Reference

A guide to Progress Embedded SQL-89 for C, including step-by-step instructions on building ESQL-89 applications and reference information on all Embedded SQL-89 Preprocessor statements and supporting function calls. This guide also describes the relationship between ESQL-89 and the ANSI standards upon which it is based.

Progress Open Client Developer's Guide

A guide that describes how to write, build, and deploy Java and ActiveX applications, and Java applets that run as clients of the Progress AppServer. This guide includes information about how to expose the AppServer as a set of Java classes or as an ActiveX server, and how to choose an Open Client distribution package for run time.

Progress SQL-89 Guide and Reference

A user guide and reference for programmers who use interactive Progress/SQL-89. It includes information on all supported SQL-89 statements, SQL-89 Data Manipulation Language components, SQL-89 Data Definition Language components, and supported Progress functions.

SQL-92

Progress Embedded SQL-92 Guide and Reference

A guide to Progress Embedded SQL-92 for C, including step-by-step instructions for building ESQL-92 applications and reference information about all Embedded SQL-92 Preprocessor statements and supporting function calls. This guide also describes the relationship between ESQL-92 and the ANSI standards upon which it is based.

Progress JDBC Driver Guide

A guide to the Java Database Connectivity (JDBC) interface and the Progress SQL-92 JDBC driver. It describes how to set up and use the driver and details the driver's support for the JDBC interface.

Progress ODBC Driver Guide

A guide to the ODBC interface and the Progress SQL-92 ODBC driver. It describes how to set up and use the driver and details the driver's support for the ODBC interface.

Progress SQL-92 Guide and Reference

A user guide and reference for programmers who use Progress SQL-92. It includes information on all supported SQL-92 statements, SQL-92 Data Manipulation Language components, SQL-92 Data Definition Language components, and Progress functions. The guide describes how to use the Progress SQL-92 Java classes and how to create and use Java stored procedures and triggers.

Deployment

Progress Client Deployment Guide

A guide that describes the client deployment process and application administration concepts and procedures.

Progress Developer's Toolkit

A guide to using the Developer's Toolkit. This guide describes the advantages and disadvantages of different strategies for deploying Progress applications and explains how you can use the Toolkit to deploy applications with your selected strategy.

Progress Portability Guide

A guide that explains how to use the Progress toolset to build applications that are portable across all supported operating systems, user interfaces, and databases, following the Progress programming model.

WebSpeed

Getting Started with WebSpeed

Provides an introduction to the WebSpeed Workshop tools for creating Web applications. It introduces you to all the components of the WebSpeed Workshop and takes you through the process of creating your own Intranet application.

WebSpeed Installation and Configuration Guide

Provides instructions for installing WebSpeed on Windows and UNIX systems. It also discusses designing WebSpeed environments, configuring WebSpeed Brokers, WebSpeed Agents, and the NameServer, and connecting to a variety of data sources.

WebSpeed Developer's Guide

Provides a complete overview of WebSpeed and the guidance necessary to develop and deploy WebSpeed applications on the Web.

WebSpeed Product Update Bulletin

A booklet that provides a brief description of each new feature of the release. The booklet also explains where to find more detailed information in the documentation set about each new feature.

Welcome to WebSpeed (Hard copy only)

A booklet that explains how WebSpeed software and media are packaged. *Welcome to WebSpeed!* also provides descriptions of the various services Progress Software Corporation offers.

Reference

Pocket Progress (Hard copy only)

A reference that lets you quickly look up information about the Progress language or programming environment.

Pocket WebSpeed (Hard copy only)

A reference that lets you quickly look up information about the SpeedScript language or the WebSpeed programming environment.

ADM2 SmartObject API Reference

The behavior of each type of SmartObject is defined by some set of procedures and functions. These routines are grouped into one or more classes of super-procedure files. Functionally, super-procedure files are comparable to dynamically-linked run-time libraries. They are loaded into memory as required by an application, and typically make their internal routines available to callers for the duration of a session.

This reference describes the set of ADM2 classes, procedures, and functions that are supplied in the current release of Progress.

For information specifically about the Progress ADM and about creating or modifying SmartObjects, see the *Progress ADM 2 Guide*. For information about creating applications using SmartObjects, see the *Progress AppBuilder Developer's Guide*.

Paths To Source Files

The default paths to the files defining the SmartObjects are:

- **Class Source Path** — %DLC%\src\adm2
- **Class Rcode Path** — %DLC%\adm2
- **Class Template Path** — %DLC%\src\adm2\template

The default paths to the ADM2 files supporting WebSpeed are:

- **Class Source Path** — %DLC%\src\web2
- **Class Rcode Path** — %DLC%\web2
- **Class Template Path** — %DLC%\src\web2\template

Objects defined under the original ADM standard (Version 8) continue to be supported. You can find their files by changing adm2 to adm (or web2 to web) in the paths shown.

Source-file Types

Each ADM class is defined by a group of ten or more files. Some of the files define the base class as distributed by Progress. Others—they have the substring "custom" in their names—are supplied as a convenience to you and stored in `src\adm2\custom`. Use those when you write custom extensions or modifications. By doing so, you reduce the risk of accidently destroying your work when you update your Progress distribution.

The naming convention for all files is: classname+filetype.extension. The classname portion can be spelled out in one filename but abbreviated in another within the same group, usually depending on total length. There is no convention for these abbreviations.

The standard files and their roles are:

- ***classname*.cld** — The class-definition file. It lists, in a comment, the class category, derivation, and the names of the class files.
- ***classname*.i, *classname*custom.i** — The primary and custom include-files. Unlike similar files in other languages such as C, these include-files sometimes have whole executable routines defined in them.
- ***classname*.p, *classname*custom.p** — The primary and custom super-procedure files. These files contain the source code for most of the routines that define the class.
- ***classname*prop.i, *classname*propcustom.i** — The primary and custom property-definition files.
- ***classname*prto.i, *classname*prtocustom.i** — The primary and custom prototype files. These files contain prototype definitions for the routines in the super-procedure files (`IN SUPER`). Prototype definitions are not the same as `FORWARD` declarations.
- ***classname*.w** — The template file.
- ***classname*exclcustom.i** — The (custom) exclusions file. This file defines `EXCLUDE-`*identifier* preprocessor variables that exclude routines from the super procedure.
- ***classname*defscustom.i** — The (custom) instance-definitions file. This file deals with custom instance properties and defines the corresponding instance-property dialog box.

For additional information, see the *Progress ADM 2 Guide*.

ADM Class Hierarchy

Figure 1 illustrates the hierarchy of relationships between the ADM classes. The placement on different levels of classes having the same parent (Container and Panel, for example) is not significant.

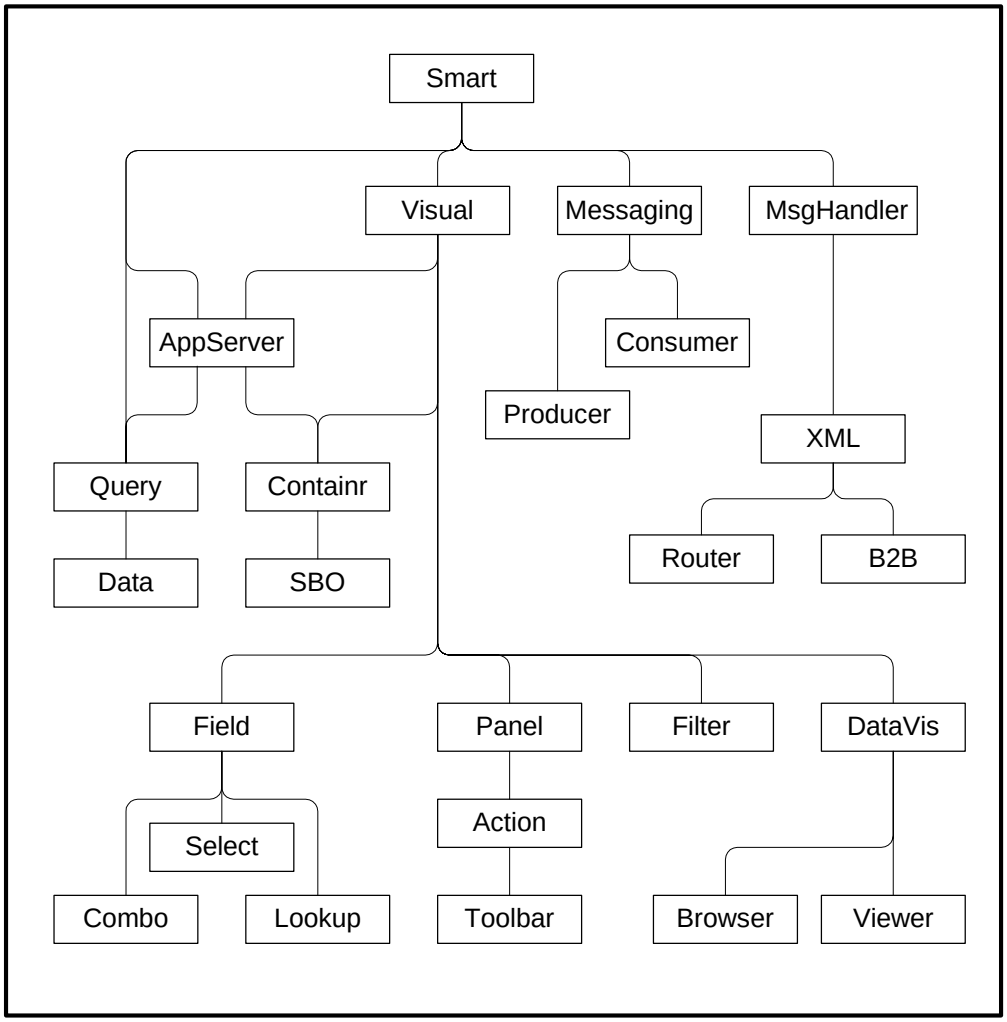

Figure 1: ADM2 Class Tree Diagram Showing Inheritance Relationships

Note that the class AppServer appears conditionally in both the Data and SBO subtrees. The switch that controls this is the AppServer-Aware check box in Procedure Settings.

SmartObjects, Templates, and Super-procedure Hierarchies

Table 1 lists in summary form the current set of SmartObjects, the groups of super-procedure files that define their capabilities, and the template files from which finished objects are created.

Table 1: SmartObjects and Their Files *(1 of 3)*

SmartObject	Super-procedure Files	Template Files
Simple SmartObject	smart.p	smart.w
Simple SmartContainer	smart.p containr.p	cntnrsimpl.w
SmartBusinessObject	smart.p visual.p (appserver.p) containr.p sbo.p sboext.p	sbo.w
SmartB2BObject	smart.p msghandler.p xml.p b2b.p	b2b.w dynb2b.w
SmartCombo	smart.p visual.p field.p combo.p	dyncombo.w
SmartConsumer (complements SmartProducer)	smart.p messaging.p consumer.p	consumer.w dynconsumer.w
SmartDataBrowser	smart.p visual.p datavis.p browser.p	browser.w dynbrowser.w
SmartDataField	smart.p visual.p field.p	field.w
SmartDataObject	smart.p (appserver.p) query.p queryext.p data.p dataext.p	data.w dyndata.w

Table 1: SmartObjects and Their Files *(2 of 3)*

SmartObject	Super-procedure Files	Template Files
SmartDataViewer	smart.p visual.p datavis.p viewer.p	viewer.w
SmartDialog	smart.p visual.p containr.p	cntnrdlg.w
SmartFilter	smart.p visual.p filter.p	dynfilter.w
SmartFolder	smart.p visual.p datavis.p	folder.w
SmartFrame	smart.p visual.p containr.p	cntnrfrm.w
SmartLookup	smart.p visual.p field.p lookup.p	dynlookup.w
SmartPanel	smart.p visual.p panel.p action.p	pcommit.w pnavico.w pnavlbl.w pupdsav.w
SmartProducer (complements SmartConsumer)	smart.p messaging.p producer.p	dynproducer.w
SmartReceiver (complements SmartSender)	smart.p messaging.p producer.p	receiver.w
SmartRouter	smart.p msghandler.p xml.p router.p	dynrouter.w

Table 1: SmartObjects and Their Files *(3 of 3)*

SmartObject	Super-procedure Files	Template Files
SmartSelect (special case of SmartDataField)	smart.p visual.p field.p select.p	dynselect.w
SmartSender	smart.p messaging.p producer.p	sender.w
SmartToolbar	smart.p visual.p panel.p action.p toolbar.p	dyntoolbar.w

SmartObjects and Their Files

This section briefly describes each of the SmartObjects in the current distribution and lists all the files that define it.

Simple SmartObject

The Simple SmartObject is essentially a template you can use when defining other objects. It has no special class identity of its own; it is an expression of the class Smart.

The file %DLC%/src/adm2/smart.p is the super procedure supporting the unimproved Simple SmartObject. The syntax for loading this file is:

SYNTAX

```
RUN start-super-proc("adm2/smart.p":U).
```

Files related to `smart.p` are:

Class Files		Custom Files	
Definition	`smart.cld`	**Super**	`smartcustom.p`
Method	`smart.i`	**Method**	`smartcustom.i`
Property	`smrtprop.i`	**Property**	`smrtpropcustom.i`
Prototype	`smrtprto.i`	**Prototype**	`smrtprtocustom.i`
Template	`smart.w`	**Exclude**	`smartexclcustom.i`
Other	`admmsgs.i`	**Instance**	`smartdefscustom.i`

Simple SmartContainer

The Simple SmartContainer provides all functionality from the class Container without the overhead of a visible run-time representation. It has no special class or super-procedure file of its own; it is an expression of the Container class and uses `containr.p` as its super-procedure file. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/containr.p":U).
```

Other files related to `containr.p` and the Simple SmartObject are:

Class Files		Custom Files	
Definition	`containr.cld`	**Super**	`containrcustom.p`
Method	`containr.i`	**Method**	`containrcustom.i`
Property	`cntnprop.i`	**Property**	`cntnpropcustom.i`
Prototype	`cntnprto.i`	**Prototype**	`cntnprtocustom.i`
Template	`cntnrsimpl.w`	**Exclude**	`containrexclcustom.i`
–	–	**Instance**	`containrdefscustom.i`

AppServer Class

If the macro {&APP-SERVER-VARS} is defined, the Container and Query classes will inherit from the AppServer class, represented by the super procedure appserver.p. To force the macro to be defined, set the AppServer-Aware check box in the Procedure Settings dialog box for the object. If there is no such check box, or it is disabled, then you cannot force the definition. The syntax for loading appserver.p is:

SYNTAX

```
RUN start-super-proc("adm2/appserver.p":U).
```

Files related to appserver.p are:

Class Files		Custom Files	
Definition	appserver.cld	**Super**	appservercustom.p
Method	appserver.i	**Method**	appservercustom.i
Property	appsprop.i	**Property**	appspropcustom.i
Prototype	appsprto.i	**Prototype**	appsprtocustom.i
Template	–	**Exclude**	appserverexclcustom.i
–	–	**Instance**	appserverdefscustom.i

SmartBusinessObject

The SmartBusinessObject integrates up to 20 SmartDataObjects. It is a special-purpose organizer object and a member of the class Container. SmartBusinessObjects provide a single point of contact for other objects, and allow you to synchronize updates on multiple SmartDataObjects in a single server-side transaction.

The class name of the object is SBO. It is a user-defined class. The file %DLC%/src/adm2/sbo.p is the super procedure file for the SBO class. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/sbo.p":U).
```

Other files related to `sbo.p` are:

Class Files		Custom Files	
Definition	`sbo.cld`	**Super**	`sbocustom.p`
Method	`sbo.i`	**Method**	`sbocustom.i`
Property	`sboprop.i`	**Property**	`sbopropcustom.i`
Prototype	`sboprto.i`	**Prototype**	`sboprtocustom.i`
Template	`sbo.w`	**Exclude**	`sboexclcustom.i`
Overflow File	`sboext.p`	**Instance**	`sbodefscustom.i`

Additional Method Files	`updtabledefs.i`
	`updparam.i`
	`updtablecase.i`
	`rupdflds.i`
		–	–

SmartB2BObject

The SmartB2BObject transforms data between XML and 4GL based on an agreed-upon XML schema mapped to local data representations. It performs this service on behalf of other SmartObjects, particularly the SmartBusinessObject and SmartDataObject.

A single instance transforms either inbound or outbound messages, but not both. Two instances of this object are required when transforming both inbound and outbound messages.

The class name of the object is B2B, and it inherits from classes MsgHandler and XML. All are user-defined classes.

The file %DLC%/src/`adm2/b2b.p` is the super procedure for the SmartB2BObject class. It contains logic that uses the XML mapping schema to read or store data in data objects. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/b2b.p":U).
```

Other files related to `b2b.p` are:

Class Files		Custom Files	
Definition	`b2b.cld`	**Super**	`b2bcustom.p`
Method	`b2b.i`	**Method**	`b2bcustom.i`
Property	`b2bprop.i`	**Property**	`b2bpropcustom.i`
Prototype	`b2bprto.i`	**Prototype**	`b2bprtocustom.i`
Template	`b2b.w` `dynb2b.w`	**Exclude**	`b2bexclcustom.i`
–	–	**Instance**	`b2bdefscustom.i`

MsgHandler Class

The MsgHandler class is represented by the super procedure `msghandler.p`. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/msghandler.p":U).
```

Other files related to `msghandler.p` are:

Class Files		Custom Files	
Definition	`msghandler.cld`	**Super**	`msghandlercustom.p`
Method	`msghandler.i`	**Method**	`msghandlercustom.i`
Property	`msghprop.i`	**Property**	`msghpropcustom.i`
Prototype	`msghprto.i`	**Prototype**	`msghprtocustom.i`
Template	`msghandler.w`	**Exclude**	`msghandlerexclcustom.i`
–	–	**Instance**	`msghandlerdefscustom.i`

XML Class

Xml.p presents a simplified DOM API by encapsulating the 4GL DOM statements so that x-noderef handles never need to be exposed to b2b.p. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/xml.p":U).
```

Other files related to xml.p are:

Class Files		Custom Files	
Definition	xml.cld	**Super**	xmlcustom.p
Method	xml.i	**Method**	xmlcustom.i
Property	xmlprop.i	**Property**	xmlpropcustom.i
Prototype	xmlprto.i	**Prototype**	xmlprtocustom.i
Template	xml.w	**Exclude**	xmlexclcustom.i
–	–	**Instance**	xmldefscustom.i

SmartCombo

The SmartCombo extends ADM2 Smart technology to the Combo Box level.

The class Combo is a user-defined class. The syntax for loading the super-procedure file is:

SYNTAX

```
RUN start-super-proc("adm2/combo.p":U).
```

Other files related to `combo.p` are:

Class Files		Custom Files	
Definition	`combo.cld`	**Super**	`combocustom.p`
Method	`combo.i`	**Method**	`combocustom.i`
Property	`combprop.i`	**Property**	`combopropcustom.i`
Prototype	`combprto.i`	**Prototype**	`comboprtocustom.i`
Template	`dyncombo.w`	**Exclude**	`comboexclcustom.i`
–	–	**Instance**	`combodefscustom.i`

SmartConsumer

See the “SmartProducer, SmartConsumer”section.

SmartDataBrowser

The SmartDataBrowser displays multiple virtual records in a simple, tabular row/column format. It obtains and updates the data in cooperation with a SmartDataObject or SmartBusinessObject. This object is supplied in both dynamic and customizable versions.

The class name of the object is Browser. It is a Progress class, and the file %DLC%/src/`adm2/sbo.p` is its super procedure. Browser inherits from Progress class DataVis. The syntax for loading `sbo.p` is:

SYNTAX

```
RUN start-super-proc("adm2/browser.p":U).
```

Other files related to `browser.p` are:

Class Files		Custom Files	
Definition	`browser.cld`	**Super**	`browsercustom.p`
Method	`browser.i`	**Method**	`browsercustom.i`
Property	`brsprop.i`	**Property**	`brspropcustom.i`
Prototype	`brsprto.i`	**Prototype**	`brsprtocustom.i`
Template	`browser.w` `dynbrowser.w`	**Exclude**	`browserexclcustom.i`

Additional Method Files	`brschnge.i`
	`brsend.i`
	`brsentry.i`
	`brshome.i`
	`brsleave.i`
	`brsoffhm.i`
	`brsoffnd.i`
	`brsscrol.i`
		Instance	`browserdefscustom.i`

SmartDataField

The SmartDataField brings Smart technology down to the field level. You can create a SmartDataField object using any visualization you desire, and insert it as a replacement for one of the standard Fill-ins that make up a SmartDataViewer.

The class name of the object is Field. It is a Progress class, and the file %DLC%/src/`adm2/field.p` is its super procedure. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/field.p":U).
```

Other files related to `field.p` are:

Class Files		Custom Files	
Definition	`field.cld`	**Super**	`fieldcustom.p`
Method	`field.i`	**Method**	`fieldcustom.i`
Property	`fieldprop.i`	**Property**	`fieldpropcustom.i`
Prototype	`fieldprto.i`	**Prototype**	`fieldprtocustom.i`
Template	`field.w`	**Exclude**	`fieldexclcustom.i`
–	–	**Instance**	`fielddefscustom.i`

SmartDataObject

The SmartDataObject is a data pump. It creates and manages a data stream based on the terms of a query that you define within it. You can often avoid complicated JOINs by linking SmartDataObjects together, each managing a single table. If your application will run in a distributed environment, you can update multiple SmartDataObjects on an AppServer by linking them inside a SmartBusinessObject.

The class name of the SmartDataObject is Data, and it inherits from class Query. Both are Progress classes, and the file %DLC%/src/`adm2/data.p` is Data's super procedure. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/data.p":U).
```

Other files related to `data.p` are:

Class Files		Custom Files	
Definition	`data.cld`	**Super**	`datacustom.p`
Method	`data.i`	**Method**	`datacustom.i`
Property	`dataprop.i`	**Property**	`datapropcustom.i`
Prototype	`dataprto.i`	**Prototype**	`dataprtocustom.i`
Template	`data.w` dyndata.w	**Exclude**	`dataexclcustom.i`
Overflow Files	`dataext.p` `dataextcols.p`	**Instance**	`datadefscustom.i`
Additional Method Files	`cltorsvr.i` `robjflds.i`	–	–

Query Class

The super-procedure file for class Query is `query.p`. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/query.p":U).
```

Other files related to `query.p` are:

Class Files		Custom Files	
Definition	`query.cld`	**Super**	`querycustom.p`
Method	`query.i`	**Method**	`querycustom.i`
Property	`qryprop.i`	**Property**	`qrypropcustom.i`
Prototype	`qryprto.i`	**Prototype**	`qryprtocustom.i`
Template	–	**Exclude**	`queryexclcustom.i`
Overflow Files	`queryext.p`	**Instance**	`querydefscustom.i`
Additional Method Files	`delrecst.i` `tblprep.i`	–	–

NOTE: If the macro `{&APP-SERVER-VARS}` is defined, the Query class will inherit from the AppServer class, represented by the super procedure `appserver.p`. To force the macro to be defined, set the AppServer-Aware check box in the Procedure Settings dialog box for your object. If there is no such check box, or it is disabled, then you cannot force the definition in that instance.

For the list of files defining the AppServer class, see the "AppServer Class" section.

SmartDataViewer

The SmartDataViewer displays a single virtual record at a time using a combination of basic fill-ins and, if you choose, SmartDataFields. It obtains and updates the data in cooperation with a SmartDataObject or SmartBusinessObject.

Its class is Viewer, a Progress class, for which the file %DLC%/src/`adm2/data.p` is its super procedure. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/viewer.p":U).
```

Other files related to `viewer.p` are:

Class Files		Custom Files	
Definition	`viewer.cld`	**Super**	`viewercustom.p`
Method	`viewer.i`	**Method**	`viewercustom.i`
Property	`viewprop.i`	**Property**	`viewpropcustom.i`
Prototype	`viewprto.i`	**Prototype**	`viewprtocustom.i`
Template	`viewer.w`	**Exclude**	`viewerexclcustom.i`
–	–	**Instance**	`viewerdefscustom.i`

SmartDialog

The SmartDialog is a special type of Frame object supported by a dedicated Window. One of several expressions of the class Container, it is a modal object. Because modal objects completely own the focus while open, they are best used to capture data without which the application cannot continue.

The SmartDialog has no class of its own; the file `%DLC%/src/adm2/containr.p` is its super procedure. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/containr.p":U).
```

Other files related to `containr.p` and the SmartDialog are:

Class Files		Custom Files	
Definition	–	**Super**	`containrcustom.p`
Method	`containr.i`	**Method**	`containrcustom.i`
Property	`cntnprop.i`	**Property**	`cntnpropcustom.i`
Prototype	`cntnprto.i`	**Prototype**	`cntnprtocustom.i`
Template	`cntnrdlg.w`	**Exclude**	`containrexclcustom.i`
Additional Method File	`dialogmn.i`	**Instance**	`containrdefscustom.i`

SmartFilter

The SmartFilter allows the user to reduce a data stream to a more manageable size, in real time. Logically, SmartFilter resides between some SmartDataObject and some visualization object such as a SmartDataBrowser. By choosing different setups for the Filter, you can give the eventual user more or less control over the contents of the data stream being displayed.

This object's class is Filter, a Progress class, and its super-procedure file is `%DLC%/src/adm2/filter.p`. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/filter.p":U).
```

Other files related to `filter.p` are:

Class Files		Custom Files	
Definition	`filter.cld`	**Super**	`filtercustom.p`
Method	`filter.i`	**Method**	`filtercustom.i`
Property	`filtprop.i`	**Property**	`filtpropcustom.i`
Prototype	`filtprto.i`	**Prototype**	`filtprtocustom.i`
Template	`dynfilter.w`	**Exclude**	`filterexclcustom.i`
–	–	**Instance**	`filterdefscustom.i`

SmartFolder

The SmartFolder implements a version of the now-standard tabbed-folders metaphor. SmartFolder is an expression of the Visual class, and has only a template file of its own. The super-procedure file for the Visual class is `%DLC%/src/adm2/visual.p`. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/visual.p":U).
```

Other files related to `visual.p` are:

Class Files		Custom Files	
Definition	–	**Super**	–
Method	–	**Method**	–
Property	–	**Property**	–
Prototype	–	**Prototype**	–
Template	`folder.w`	**Exclude**	–
–	–	**Instance**	–

SmartFrame

The SmartFrame provides a platform for constructing reusable subsystems. Like the SmartDialog and the SmartWindow, the SmartFrame has no special class of its own: it is an expression of the Container class, for which the super-procedure file is `%DLC%/src/adm2/containr.p`. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/containr.p":U).
```

Other files related to `container.p` are:

Class Files		Custom Files	
Definition	–	**Super**	–
Method	–	**Method**	–
Property	–	**Property**	–
Prototype	–	**Prototype**	–
Template	`cntnrfrm.w`	**Exclude**	–
–	–	**Instance**	–

SmartLookup

The SmartLookup is a faster but less-general version of the SmartSelect. It provides quick, read-only lookup using a focused, dynamic query.

The class Lookup is a user-defined class. The syntax for loading the super-procedure file is:

SYNTAX

```
RUN start-super-proc("adm2/lookup.p":U).
```

Other files related to `lookup.p` are:

Class Files		Custom Files	
Definition	`lookup.cld`	**Super**	`lookupcustom.p`
Method	`lookup.i`	**Method**	`lookupcustom.i`
Property	`lookprop.i`	**Property**	`lookuppropcustom.i`
Prototype	`lookprto.i`	**Prototype**	`lookupprtocustom.i`
Template	`dynlookup.w`	**Exclude**	`lookupexclcustom.i`
–	–	**Instance**	`lookupdefscustom.i`

SmartPanel

The SmartPanel presents an array of related buttons. Several such arrays, dedicated to different purposes, are supplied with AppBuilder. These arrays are all members of the Progress class Panel. Panel inherits from class Visual, and its super-procedure file is `%DLC%/src/adm2/panel.p`. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/panel.p":U).
```

Other files related to panel.p are:

Class Files		Custom Files	
Definition	panel.cld	**Super**	panelcustom.p
Method	panel.i	**Method**	panelcustom.i
Property	panlprop.i	**Property**	panlpropcustom.i
Prototype	panlprto.i	**Prototype**	panlprtocustom.i
Template	pcommit.w pnavico.w pnavlbl.w pupdsav.w	**Exclude**	panelexclcustom.i
–	–	**Instance**	paneldefscustom.i

SmartProducer, SmartConsumer

The SmartMessageProducer sends messages using some message-transport system. At present, the only transport system supported is SonicMQ.

On demand, SmartMessageProducer creates a message body of the appropriate kind, passes it back to the requesting object—such as a SmartB2BObject or SmartSender—to be filled in, and finally inserts the message into the outbound message-transport queue. If it receives a reply to a message, it accepts it and passes it upstream for processing.

The SmartConsumer handles inbound traffic from some message-transport system. It accepts incoming messages and passes them on for processing by some other object such as a SmartB2BObject or SmartReceiver. It also sends reply messages when required.

The class names of the objects are Producer and Consumer, respectively. They are Progress classes derived from the Messaging class.

The file %DLC%/src/adm2/producer.p is the super procedure for the class SmartProducer. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/producer.p":U).
```

The file %DLC%/src/adm2/consumer.p is the super procedure for the class SmartConsumer. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/consumer.p":U).
```

Other files related to producer.p are:

Class Files		Custom Files	
Definition	producer.cld	**Super**	producercustom.p
Method	producer.i	**Method**	producercustom.i
Property	prodprop.i	**Property**	prodpropcustom.i
Prototype	prodprto.i	**Prototype**	prodprtocustom.i
Template	producer.w dynproducer.w	**Exclude**	producerexclcustom.i
–	–	**Instance**	producerdefscustom.i

The corresponding files for the Consumer class are:

Class Files		Custom Files	
Definition	consumer.cld	**Super**	consumercustom.p
Method	consumer.i	**Method**	consumercustom.i
Property	consprop.i	**Property**	conspropcustom.i
Prototype	consprto.i	**Prototype**	consprtocustom.i
Template	consumer.w dynconsumer.w	**Exclude**	consumerexclcustom.i
–	–	**Instance**	consumerdefscustom.i

Messaging Class

The file %DLC%/src/adm2/messaging.p is the super procedure for the class Messaging. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/messaging.p":U).
```

Other files related to messaging.p are:

Class Files		Custom Files	
Definition	messaging.cld	**Super**	messagingcustom.p
Method	messaging.i	**Method**	messagingcustom.i
Property	messprop.i	**Property**	messpropcustom.i
Prototype	messprto.i	**Prototype**	messprtocustom.i
Template	–	**Exclude**	messagingexclcustom.i
–	–	**Instance**	messagingdefscustom.i

SmartReceiver

See the “SmartSender, SmartReceiver” section.

SmartRouter

The SmartRouter is a utility object. Its role is to route incoming documents to the appropriate SmartB2BObject for transformation. Its class is Router, a user-defined class whose super-procedure file is %DLC%/src/adm2/router.p. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/router.p":U).
```

Other files related to `router.p` are:

Class Files		Custom Files	
Definition	`router.cld`	**Super**	`routercustom.p`
Method	`router.i`	**Method**	`routercustom.i`
Property	`routprop.i`	**Property**	`routpropcustom.i`
Prototype	`routprto.i`	**Prototype**	`routprtocustom.i`
Template	`router.w`	**Exclude**	`routerexclcustom.i`
–	–	**Instance**	`routerdefscustom.i`

SmartSelect

The SmartSelect object is a type of SmartDataField. It represents a self-populating Selection List. Its class is a Progress-type class, Select, whose super-procedure file is `%DLC%/src/adm2/select.p`. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/select.p":U).
```

Other files related to `select.p` are:

Class Files		Custom Files	
Definition	`select.cld`	**Super**	`selectcustom.p`
Method	`select.i`	**Method**	`selectcustom.i`
Property	`seleprop.i`	**Property**	`selepropcustom.i`
Prototype	`seleprto.i`	**Prototype**	`seleprtocustom.i`
Template	`select.w`	**Exclude**	`selectexclcustom.i`
–	–	**Instance**	`selectdefscustom.i`

SmartSender, SmartReceiver

These message-handling objects can usually be substituted for the SmartB2BObject when you do not need to perform protocol-based transformation between XML and 4GL. To use these objects, you must complete their handler routines to suit your customer's business needs. You can make the handler functions as simple or complex as you like.

The SmartSender and SmartReceiver objects are based on the msghandler class, whose super procedure is `msghandler.p`. The Msghandler class is derived from the class Smart. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/msghandler.p":U).
```

Other files related to `msghandler.p` are:

Definition	`msghandler.cld`	**Super**	`msghandlercustom.p`
Method	`msghandler.i`	**Method**	`msghandlercustom.i`
Property	`msghprop.i`	**Property**	`msghpropcustom.i`
Prototype	`msghprto.i`	**Prototype**	`msghprtocustom.i`
Template	`receiver.w` `sender.w`	**Exclude**	`msghandlerexclcustom.i`
–	–	**Instance**	`msghandlerdefscustom.i`

SmartToolbar

The SmartToolbar object is related to the SmartPanel, combining a menu component with a tool bar. The default SmartToolbar can replace all the dedicated SmartPanels that are also distributed with AppBuilder.

Both the menu and the toolbar can be turned off. If you turn off the toolbar, you can use SmartPanels to supply the same capabilities. If you turn off the menu, however, you have no menu at all. You cannot use the menu that is available through the SmartWindow properties dialog box. The two menu designs are not compatible.

The SmartToolbar is an expression of the Progress Toolbar class, descended from the Panel class through the Actions class. The Toolbar super-procedure file is `%DLC%/src/adm2/toolbar.p`. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/toolbar.p":U).
```

Other files related to `toolbar.p` are:

Class Files		Custom Files	
Definition	`toolbar.cld`	**Super**	`toolbarcustom.p`
Method	`toolbar.i`	**Method**	`toolbarcustom.i`
Property	`toolprop.i`	**Property**	`toolpropcustom.i`
Prototype	`toolprto.i`	**Prototype**	`toolprtocustom.i`
Template	`toolbar.w` dyntoolbar.w	**Exclude**	`toolbarexclcustom.i`
Overflow Files	`toolbarext.p`	**Instance**	`toolbardefscustom.i`

Action Class

The super-procedure file for the Action class is %DLC%/src/adm2/action.p. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/action.p":U).
```

Other files related to `action.p` are:

Class Files		Custom Files	
Definition	`action.cld`	**Super**	`actioncustom.p`
Method	`action.i`	**Method**	`actioncustom.i`
Property	`actiprop.i`	**Property**	`actipropcustom.i`
Prototype	`actiprto.i`	**Prototype**	`actiprtocustom.i`
Template	`action.w`	**Exclude**	`actionexclcustom.i`
–	–	**Instance**	`actiondefscustom.i`

SmartWindow

The SmartWindow object is the leading member of the Container class. Unlike other members of that class, SmartWindows are extremely general and versatile. SmartWindow, like SmartFrame and SmartDialog, does not have a special class of its own. It is an expression of the Container class. The super-procedure file for the Container class is `%DLC%/src/adm2/containr.p`. The syntax for loading that file is:

SYNTAX

```
RUN start-super-proc("adm2/containr.p":U).
```

Other files related to `container.p` are:

Class Files		Custom Files	
Definition	`containr.cld`	**Super**	`containrcustom.p`
Method	`containr.i`	**Method**	`containrcustom.i`
Property	`cntnprop.i`	**Property**	`cntnpropcustom.i`
Prototype	`cntnprto.i`	**Prototype**	`cntnprtocustom.i`
Template	`cntnrwin.w`	**Exclude**	`containrexclcustom.i`
Additional Method File	`windowmn.i`	**Instance**	`containrdefscustom.i`

Reading and Writing Object Properties

Every SmartObject makes its public properties available to other objects. The other objects might be other SmartObjects, non-Smart 4GL code, or non-Progress modules using the Open4GL interface.

The standard properties for an object type are defined in the ADMProps temp-table. That table and its global handle `ghADMProps` are declared in the file `smrtprop.i`. Fields in the table represent individual properties, and are defined by the class that needs them. For example, a SmartDataObject has properties defined in `dataprop.i`, `qryprop.i`, and `smrtprop.i`—`smrtprop.i` declares the ADMProps table and defines properties common to all SmartObjects, `qryprop.i` adds properties common to all query-based objects, and finally `dataprop.i` adds the properties specific to the SmartDataObject.

For additional information, see the *Progress ADM 2 Guide*.

Refer to Figure 1 to see the chain of inheritance for each object class.

get*propname* and set*propname* Functions

Every public-readable property must have a function defined to return the current value of the property. Similarly, every public-writable property must have a function defined to set a new value for the property. These functions conform to certain conventions:

- **get** — The get function identifier takes the form `get`*`propname`*. The function accepts no arguments, and returns the current value of the property. It can also perform other processing, if needed.
- **set** — The set function identifier takes the form `set`*`propname`*. It accepts a single argument—the new value for the property—and returns TRUE/FALSE depending on whether it succeeded in changing the value. It can perform additional processing, if needed.

A very small number of these functions vary slightly from the model described here, but these differences are not significant.

{get} and {set} Pseudo-functions

In addition to the conventional get and set functions, the ADM supports a pseudo-function syntax for use in super-procedure files particularly. These pseudo-functions are implemented by the include files `get` and `set` (**not** `get.i` or `set.i`) located in the tty and gui directories.

A call to {get} takes one of the following forms:

```
{get propname returnvar}.
{get propname returnvar ownerhandle}
```

Where *propname* is the property name in the TARGET-PROCEDURE object and *returnvar* is the variable into which the value is to be returned. If the property does not belong to the TARGET-PROCEDURE object, provide the appropriate handle in *ownerhandle*.

The {set} syntax is very similar:

```
{set propname newvalue}
{set propname newvalue ownerhandle}
```

The only difference between {get} and {set} syntax is that the second {set} argument is the new value for the property.

The Need For Two Different Syntaxes

Properties store the current state of the object, so reading and writing them is a high-frequency activity. It pays to optimize high-frequency activities, even when the net gain per operation is small. The pseudo-functions provide that slight optimization, and should be used in code that runs inside a SmartObject. Code that runs outside the Smart world—basic 4GL and Open4GL routines—should use the conventional get/set functions.

Figure 2 shows the {get} code reformatted and commented. The {set} code is very similar.

```
/* {get propname returnvar [ownerhandle]} */
&IF "{3}":U = "":U &THEN                /* if no handle is passed in... */
   &IF DEFINED(xp{1}) NE 0 &THEN         /* but there is an xp macro defined */
      ASSIGN
         ghProp = WIDGET-HANDLE(             /* pick up the handle to the */
          ENTRY(1,                         /* target proc's ADM-DATA table */
          TARGET-PROCEDURE:ADM-DATA,
           CHR(1)))
         ghProp = ghProp:BUFFER-FIELD('{1}':U)  /* then pick up the handle */
                                             /* to the slot for the prop */
         {2} = ghProp:BUFFER-VALUE           /* and finally assign the */
                                             /* value in the slot to */
                                             /* the prop var passed in */
&ELSE                                     /* and there is no xp macro defined */
      {2} = dynamic-function(
            "get{1}":U IN TARGET-PROCEDURE)  /* set the prop var by */
                                             /* building and firing a call */
                                             /* to the ordinary function */
   &ENDIF                                   /* end xp-macro-defined */
&ELSE                                     /* a handle was passed in... */
   &IF DEFINED(xp{1}) NE 0 &THEN            /* and if there is an xp macro */
      ASSIGN
        ghProp = WIDGET-HANDLE(             /* pick up the handle to */
                ENTRY(1,                    /* the passed-in handle's */
                {3}:ADM-DATA,               /* props table */
                 CHR(1)))
        ghProp = ghProp:BUFFER-FIELD('{1}':U)  /* pick up the handle to */
                                            /* the slot for the prop in */
                                               /* that table */
        {2} = ghProp:BUFFER-VALUE           /* and finally set the variable */
                                               /* with contents of that slot */
   &ELSE                                    /* but no xp macro defined */
      {2} = dynamic-function(               /* set the var by building */
             "get{1}":U IN {3})             /* and firing a call to the */
                                               /* ordinary function */
   &ENDIF                                  /* end if-xp-macro-defined */
&ENDIF                                   /* end if-no-handle-passed-in */
```

Figure 2: Commented Source Code Of GET Pseudo-function

List Of Public Properties By Name

Except as noted, these properties can all be read/written using either the corresponding get/set function or the {get}/{set} pseudo-function syntax.

NOTE: At present, the files `actiprop.i` and `viewprop.i` have no Progress-defined properties in them. The file `desiprop.i` is a template properties file, supplied as a convenience to you; you can use it when defining a new type of object.

The last four columns in each table are described as follows:

- **X** — If marked, an xp*propname* token has been defined, allowing use of the {get}/{set} pseudo-functions. This syntax should be used from within SmartObjects.
- **G** — If marked, a conventional get*propname* function has been defined. Use these functions in non-Smart code.
- **S** — If marked, a conventional set*propname* function has been defined. Use these functions in non-Smart code.
- **I** — If checked, this is an instance property.

Table 2: Public Properties By Name: A *(1 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
ActionEvent	CHARACTER	brsprop.i	–	X	X	–
ActionGroups	CHARACTER	toolprop.i	X	X	X	X
AddFunction [1], [2]	CHARACTER	panlprop.i	X	X	X	X
ApplyActionOnExit	LOGICAL	brsprop.i	X	X	X	–
ApplyExitOnAction	LOGICAL	brsprop.i	X	X	X	–
AppService [3]	CHARACTER	appsprop.i	-	X	X	X
ASDivision [4], [5]	CHARACTER	appsprop.i	X	X	X	–
ASHandle [6]	HANDLE	dataprop.i	–	X	X	–
ASHandle	HANDLE	sboprop.i	-	-	-	–

Table 2: Public Properties By Name: A *(2 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
ASHasStarted	LOGICAL	appsprop.i	X	X	X	-
ASInfo [7]	CHARACTER	appsprop.i	X	X	X	X
ASInitializeOnRun	LOGICAL	appsprop.i	X	X	X	-
AssignList [8]	CHARACTER	qryprop.i	X	X	-	-
ASUsePrompt [9]	LOGICAL	appsprop.i	X	X	X	X
AutoCommit [10], [11]	LOGICAL	dataprop.i	X	X	X	-
AutoCommit [12]	LOGICAL	sboprop.i	X	X	X	-
AutoRefresh [13]	LOGICAL	seleprop.i	X	X	X	X
AvailMenuActions [14]	CHARACTER	toolprop.i	X	X	X	-
AvailToolbarActions [14]	CHARACTER	toolprop.i	X	X	X	-

1 Initial value "One-Record".

2 The set function is defined in `pupdsav.w`.

3 Because the SBO (SmartBusinessObject) is designed to integrate many SDOs (SmartDataObjects), most SDO properties have counterparts in the SBO.

4 The Division property. Possible values are "Client", "Server", "Both", or the empty string.

5 Identifies where the caller is running. Possible values are "Client", "Server", or the empty string meaning the application is not distributed.

6 The handle of the copy running on the AppServer. This is the persistent procedure handle, used when invoking procedures and functions.

7 The string, if any, used as a parameter during connection to the AppService.

8 The list of updatable columns whose names have been modified in the SDO.

9 TRUE if the supporting code should prompt for a username/password when connecting to the AppService.

10 TRUE if updates automatically commit. If the SDO is in an SBO, this property is set to FALSE.

11 A set function exists in `query.p` also, though the property is not declared in `qryprop.i`. That set function is overridden by the set function in `dataext.p`.

[12] Defaults to FALSE. When set to TRUE, updates automatically commit.

[13] Property declared, but not currently in use.

[14] Actions available in the calling Toolbar object. This list and the list in the AvailToobarActions property are displayed in the instance-properties dialog box. The selected actions are saved as ActionGroups.

Table 3: Public Properties By Name: B *(1 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
BindSignature	CHARACTER	appsprop.i	X	-	-	-
BlockDataAvailable	LOGICAL	sboprop.i	X	X	X	-
BoxRectangle	HANDLE	panlprop.i	X	X	-	-
BrowseContainer	HANDLE	seleprop.i	X	-	-	-
BrowseFields	CHARACTER	seleprop.i	X	X	X	X
BrowseHandle	HANDLE	brsprop.i	X	X	-	-
BrowseInitted	LOGICAL	brsprop.i	X	-	-	-
BrowseObject	HANDLE	seleprop.i	X	-	-	-
BrowseProcedure	CHARACTER	seleprop.i	X	-	-	-
BrowseTitle	CHARACTER	seleprop.i	X	X	X	X
BrowseWindowProcedure	CHARACTER	seleprop.i	X	-	-	-
BufferHandles	CHARACTER	qryprop.i	X	-	-	-
ButtonCount	INTEGER	panlprop.i	X	X	-	-
ButtonHandle	HANDLE	seleprop.i	X	-	-	-
BoxRectangle [1]	HANDLE	panlprop.i	X	X	X	-
BrowseContainer	HANDLE	seleprop.i	X	X	X	-
BrowseFields [2]	CHARACTER	seleprop.i	X	X	X	-
BrowseHandle	HANDLE	brsprop.i	X	X	X	-

Table 3: Public Properties By Name: B *(2 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
BrowseInitted	LOGICAL	brsprop.i	X	X	X	-
BrowseObject	HANDLE	seleprop.i	X	X	X	-
BrowseProcedure	CHARACTER	seleprop.i	X	X	X	-
BrowseTitle [2]	CHARACTER	seleprop.i	X	X	X	-
BrowseWindowProcedure	CHARACTER	seleprop.i	X	-	-	-
BufferHandles	CHARACTER	qryprop.i	X	X	X	-
ButtonCount	INTEGER	panlprop.i	X	X	X	-
ButtonHandle	HANDLE	seleprop.i	X	X	X	-

[1] The handle of the rectangle, if any, that provides the visual border of a SmartPanel.

[2] Information to be displayed when the ViewAs property is BROWSE.

Table 4: Public Properties By Name: C *(1 of 3)*

Property Name	Data Type	Declared in	X	G	S	I
CalcWidth	LOGICAL	brsprop.i	X	X	X	X
CancelBrowseOnExit [1]	LOGICAL	seleprop.i	X	X	X	X
CascadeOnBrowse [2]	LOGICAL	sboprop.i	X	X	X	X
ChangedEvent [3]	CHARACTER	seleprop.i	X	X	X	X
CheckCurrentChanged [4]	LOGICAL	dataprop.i	X	X	X	X
CheckLastOnOpen [5]	LOGICAL	qryprop.i	X	X	X	–
ClientID [6]	CHARACTER	messprop.i	X	X	X	X

Table 4: Public Properties By Name: C *(2 of 3)*

Property Name	Data Type	Declared in	X	G	S	I
CommitSource	HANDLE	dataprop.i	X	X	X	–
CommitSource	HANDLE	sboprop.i	X	X	X	–
CommitSourceEvents [7]	CHARACTER	dataprop.i	X	X	X	–
CommitSourceEvents [7]	CHARACTER	sboprop.i	X	X	–	–
CommitTarget	CHARACTER	dataprop.i	X	X	X	–
CommitTarget [8]	CHARACTER	panlprop.i	X	X	X	–
CommitTargetEvents [7]	CHARACTER	dataprop.i	X	X	X	–
CommitTargetEvents [7], [8]	CHARACTER	panlprop.i	X	X	X	–
ContainedDataColumns [9]	CHARACTER	sboprop.i	X	X	X	–
ContainedDataObjects [10]	CHARACTER	sboprop.i	X	X	–	–
ContainerHandle [11]	HANDLE	smrtprop.i	X	X	–	–
ContainerHidden	LOGICAL	smrtprop.i	X	X	X	–
ContainerSource	HANDLE	smrtprop.i	X	X	X	–
ContainerSourceEvents	CHARACTER	smrtprop.i	X	X	X	–
ContainerTarget	CHARACTER	cntnprop.i	X	X	X	–
ContainerTargetEvents	CHARACTER	cntnprop.i	X	X	–	–
ContainerType [12]	CHARACTER	smrtprop.i	X	X	–	–
CreateHandles [13]	CHARACTER	dvisprop.i	X	X	–	–

Table 4: Public Properties By Name: C *(3 of 3)*

Property Name	Data Type	Declared in	X	G	S	I
CurrentMessage	HANDLE	prodprop.i	X	X	X	–
CurrentMessageId [14]	CHARACTER	msghprop.i	X	X	X	–
CurrentPage	INTEGER	cntnprop.i	X	X	–	–

1 Set to TRUE if the value in the browse is **not** to be selected on exit. Should be TRUE when ExitBrowseOnAction is TRUE because if the user exits the browse with a value selected, the Close button can function as a Cancel.

2 Determines whether data will be retrieved from a dependent SDO when the parent SDO has more than one row in its current dataset. Defaults to TRUE.

3 Optional event to publish on value-changed. Be sure to define the corresponding SUBSCRIBE in the container.

4 TRUE if the SDO should determine whether the record being updated has changed by some other object.

5 TRUE if a get-last should be performed. See the code for detailed information.

6 Declared in `messprop.i`; defined as an instance property in `consprop.i` and `prodprop.i`.

7 The list of events in the respective SOURCE or TARGET object to which this object subscribes.

8 Separate get and set functions are defined in `pcommit.w` and in `toolbar.p`. None are defined in `panel.p`.

9 An inclusive list of all the columns being served by all the SDOs in this SBO. Columns for an SDO are comma-delimited. Columns belonging to different SDOs are separated by a semicolon. The groups of columns are in the same order as the SDOs listed in the ContainedDataObjects property.

10 A list of handles to all SDOs in this SBO.

11 Handle of the caller's parent container widget (window or frame).

12 Identifies the caller's container type. Possible values are "Window", "Frame", or the empty string if the caller is not a container.

13 A list of handles to the fields that should be enabled for an Add or Copy operation on a record.

14 The ID from the last sendMessage with ReplyRequired set.

Table 5: Public Properties By Name: D *(1 of 4)*

Property Name	Data Type	Declared in	X	G	S	I
DataColumns [1]	CHARACTER	sboprop.i	X	X	–	–
DataColumns	CHARACTER	qryprop.i	X	X	–	–
DataColumnsByTable[2]	CHARACTER	qryprop.i	X	–	–	–

Table 5: Public Properties By Name: D *(2 of 4)*

Property Name	Data Type	Declared in	X	G	S	I
DataFieldDefs	CHARACTER	dataprop.i	X	X	–	–
DataHandle	HANDLE	dataprop.i	X	X	–	–
DataIsFetched	LOGICAL	qryprop.i	X	X	X	-
DataModified	LOGICAL	dataprop.i	X	X	–	–
DataModified [4]	LOGICAL	dvisprop.i	–	X	X	–
DataModified	LOGICAL	fieldprop.i	–	X	X	–
DataModified	LOGICAL	filtprop.i	X	–	–	–
DataObject	CHARACTER	filtprop.i	X	X	X	-
DataObjectNames [5]	CHARACTER	sboprop.i	X	X	X	X
DataObjectOrdering [6]	CHARACTER	sboprop.i	X	X	–	–
DataQueryBrowsed [7]	LOGICAL	dataprop.i	–	X	X	–
DataQueryString	CHARACTER	dataprop.i	-	X	X	–
DataSource	HANDLE	smrtprop.i	-	X	X	–
DataSourceEvents [8]	CHARACTER	smrtprop.i	X	X	X	–
DataSourceFilter [9]	CHARACTER	seleprop.i	X	X	X	X
DataSourceNames	CHARACTER	smrtprop.i	X	X	X	X
DataTarget [10]	CHARACTER	smrtprop.i	X	X	X	–
DataTargetEvents [8]	CHARACTER	smrtprop.i	X	X	X	–
DataValue	CHARACTER	seleprop.i	-	X	X	-
DBAware [11]	LOGICAL	smrtprop.i	X	X	X	–

Table 5: Public Properties By Name: D *(3 of 4)*

Property Name	Data Type	Declared in	X	G	S	I
DBNames	CHARACTER	qryprop.i	-	X	X	-
DefaultCharWidth	DECIMAL	filtprop.i	X	X	X	X
DefaultEditorLines	INTEGER	filtprop.i	X	X	X	X
DefaultHeight	DECIMAL	filtprop.i	X	-	-	-
DefaultLayout	CHARACTER	visprop.i	X	X	X	-
DefaultLogical	CHARACTER	filtprop.i	X	X	X	-
DefaultWidth	DECIMAL	filtprop.i	X	X	X	X
DefineAnyKeyTrigger [12]	LOGICAL	seleprop.i	-	X	X	X
DesignDataObject	CHARACTER	smrtprop.i	X	X	X	X
Destination	CHARACTER	msghprop.i	X	X	X	-
DestinationList	CHARACTER	b2bprop.i	X	X	X	X
Destinations	CHARACTER	consprop.i	X	X	X	X
DestroyStateless	LOGICAL	dataprop.i	X	X	X	X
DirectionList	CHARACTER	b2bprop.i	X	X	X	X
DisabledActions	CHARACTER	toolprop.i	-	X	X	X
DisableOnInit	LOGICAL	visprop.i	X	X	X	X
DisconnectAppServer	LOGICAL	dataprop.i	X	X	X	X
DisplayedField [13]	CHARACTER	seleprop.i	X	X	X	X
DisplayedFields	CHARACTER	dvisprop.i	X	X	X	X
DisplayedFields [14]	CHARACTER	filtprop.i	X	X	X	X

Table 5: Public Properties By Name: D *(4 of 4)*

Property Name	Data Type	Declared in	X	G	S	I
DisplayedTables	CHARACTER	dvisprop.i	X	X	–	–
DisplayField	LOGICAL	fieldprop.i	X	X	X	X
DisplaySelection	(Undefined)	seleprop.i	X	–	–	–
DisplayValue [15]	CHARACTER	seleprop.i	–	X	X	–
DocTypeList	CHARACTER	b2bprop.i	X	X	X	X
DocumentHandle	HANDLE	xmlprop.i	X	X	X	-
Domain [16]	CHARACTER	messprop.i	X	X	X	X
DynamicSDOProcedure	CHARACTER	cntnprop.i	X	X	X	–
DTDPublicId	CHARACTER	xmlprop.i	-	X	X	-
DTDPublicIdList	CHARACTER	b2bprop.i	X	X	X	X
DTDSystemId	CHARACTER	xmlprop.i	-	X	X	
DTDSystemList	CHARACTER	b2bprop.i	X	X	X	X
DynamicSDOProcedure	CHARACTER	cntnprop.i	X	X	X	–

1 An inclusive list of all the columns being served by all the SDOs in this SBO, each qualified by the ObjectName of its SDO. Columns are comma-delimited. This is the same information stored in the ContainedDataColumns property, but organized differently.

2 Comma-delimited list of all column names in the SDO. The get/set functions have truncated names: getDataColumns/setDataColumns.

3 Returns the handle of the appropriate RowObject query.

4 Indicates whether there is an unsaved modification to some SCREEN-VALUE. A set function is also defined in `browser.p`.

5 The ordered list of ObjectNames of contained SDOs. This property is not normally changed under program control.

6 The mapping of the programmer-defined update order onto the AppBuilder-defined Update Tables.

7 TRUE if this SDO's data stream is already assigned to a browser. Essentially a lock to prevent conflicts.

8 A list of the events in the respective SOURCE or TARGET to which this object subscribes. See notes in the code header for additional details.

[9] Optional filter expression to be applied to the DATA-SOURCE.

[10] Typically used in connection with pass-through links.

[11] TRUE if the object requires a local database connection.

[12] TRUE if a persistent trigger defined on ANY-KEY. Only used for a fill-in generated for the view-as-browse option.

[13] Field being displayed in the selection

[14] Comma-separated list of the columns being displayed by the data-display object.

[15] Returns the value displayed by the SmartSelect. This is not necessarily the same as the value to be stored.

[16] Storage declared, xp token, and get/set defined in `messprop.i`, but defined as instance property in `consprop.i` and `prodprop.i`.

Table 6: Public Properties By Name: E

Property Name	Data Type	Declared in	X	G	S	I
EnabledFields[1]	CHARACTER	dvisprop.i	X	X	X	–
EnabledFields	CHARACTER	filtprop.i	X	X	X	–
EnabledHandles[2]	CHARACTER	dvisprop.i	X	X	–	–
EnabledHandles	CHARACTER	filtprop.i	X	–	–	–
EnabledObjFlds[3]	CHARACTER	visprop.i	X	X	–	–
EnabledObjHdls[4]	CHARACTER	visprop.i	X	X	–	–
EnableField	LOGICAL	fieldprop.i	X	X	X	X
EnableOnAdd	LOGICAL	seleprop.i	X	X	X	–
EnableSelection	(Undefined)	seleprop.i	X	–	–	–
ErrorConsumer	HANDLE	consprop.i	X	–	–	–
ErrorConsumer	HANDLE	prodprop.i	X	–	–	–
ExitBrowseOnAction[5]	LOGICAL	seleprop.i	X	X	X	X
ExternalRefList	CHARACTER	routprop.i	X	X	X	X

[1] Returns the list of enabled fields in the display object that have counterparts in the SDO's data stream.

[2] The handles are to the enabled fields in the display object.

[3] The fields in this list are not associated with data fields.

[4] The handles are to widgets not associated with data fields.

[5] TRUE if selecting a value in the browse exits the browse. Triggered by DEFAULT-ACTION (RETURN or double-click).

Table 7: Public Properties By Name: F *(1 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
FetchOnOpen	CHARACTER	qryprop.i	X	–	–	–
FieldColumn	DECIMAL	filtprop.i	X	X	X	X
FieldEnabled	LOGICAL	fieldprop.i	X	X	X	–
FieldFormats	CHARACTER	filtprop.i	X	X	X	X
FieldHandles	CHARACTER	dvisprop.i	X	X	–	–
FieldHandles	CHARACTER	filtprop.i	X	–	–	–
FieldHelpIds	CHARACTER	filtprop.i	X	X	X	X
FieldLabels	CHARACTER	filtprop.i	X	X	X	X
FieldName	CHARACTER	fieldprop.i	X	X	X	X
FieldOperatorStyles	CHARACTER	filtprop.i	X	X	X	X
FieldsEnabled	LOGICAL	dvisprop.i	X	X	–	–
FieldsEnabled	LOGICAL	filtprop.i	X	–	–	–
FieldTooltips	CHARACTER	filtprop.i	X	X	X	X
FieldWidths	CHARACTER	filtprop.i	X	X	X	X
FillBatchOnRepos	LOGICAL	dataprop.i	X	X	X	–
FilterSource	HANDLE	qryprop.i	X	X	X	–
FilterSource	HANDLE	cntnprop.i	X	X	X	–

Table 7: Public Properties By Name: F *(2 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
FilterTarget	CHARACTER	filtprop.i	–	X	X	–
FilterTargetEvents	CHARACTER	filtprop.i	X	X	X	–
FilterWindow	CHARACTER	qryprop.i	X	X	X	–
FirstResultRow	CHARACTER	dataprop.i	X	X	X	–
FirstRowNum[1]	INTEGER	dataprop.i	X	X	X	–
FlatButtons[2]	LOGICAL	toolprop.i	X	X	X	X
FolderFont	INTEGER	folder.w	X	X	X	X
FolderLabels	CHARACTER	folder.w	X	X	X	X
FolderTabHeight	DECIMAL	folder.w	-	X	-	X
FolderTabType	INTEGER	folder.w	X	X	X	X
FolderTabWidth	DECIMAL	folder.w	X	X	X	X
ForeignFields[3]	CHARACTER	qryprop.i	-	X	X	X
ForeignFields	CHARACTER	sboprop.i	X	X	X	X
ForeignValues[4]	CHARACTER	qryprop.i	X	X	X	–
ForeignValues	CHARACTER	sboprop.i	X	X	X	–
Format[5]	CHARACTER	seleprop.i	X	X	X	X

1 Temp-table first-row number.

2 Not currently in use.

3 Comma-separated list of pairs: local database field name, source temp-table field name, local database field name, source.... Used for opening dependent queries.

[4] CHR(1)-separated, formatted strings representing the most received foreign-field values.

[5] Overridden format, used when view-as is BROWSE and data displayed as fill-in.

Table 8: Public Properties By Name: G

Property Name	Data Type	Declared in	X	G	S	I
GroupAssignSource	HANDLE	dvisprop.i	X	X	X	–
GroupAssignSourceEvents[1]	CHARACTER	dvisprop.i	X	X	X	–
GroupAssignTarget	CHARACTER	dvisprop.i	X	X	X	–
GroupAssignTargetEvents[1]	CHARACTER	dvisprop.i	X	X	X	–

[1] Comma-separated list of events to which this object subscribes in the respective source/target.

Table 9: Public Properties By Name: H

Property Name	Data Type	Declared in	X	G	S	I
HelpId	INTEGER	seleprop.i	X	X	X	X
HideOnInit	LOGICAL	visprop.i	X	X	X	X

Table 10: Public Properties By Name: I *(1 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
ImagePath[1]	CHARACTER	toolprop.i	X	X	X	–
IndexInformation	CHARACTER	dataprop.i	–	X	X	–
InMessageSource	HANDLE	msghprop.i	X	X	X	–
InMessageTarget	HANDLE	consprop.i	X	X	X	–
InstanceProperties	CHARACTER	smrtprop.i	X	X	X	–

Table 10: Public Properties By Name: I *(2 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
InternalRefList	CHARACTER	routprop.i	X	X	X	X

[1] Path to the image within the file system.

Table 11: Public Properties By Name: J

Property Name	Data Type	Declared in	X	G	S	I
JMSpartition	CHARACTER	messprop.i	X	X	X	X
JMSpassword	CHARACTER	messprop.i	X	X	X	X
JMSPassword	CHARACTER	consprop.i	X	–	–	X
JMSuser	CHARACTER	messprop.i	X	X	X	X
JMSUser	CHARACTER	consprop.i	X	–	–	X

Table 12: Public Properties By Name: K

Property Name	Data Type	Declared in	X	G	S	I
KeyField	CHARACTER	seleprop.i	X	X	X	X
KeyFields	CHARACTER	qryprop.i	X	–	–	–
KeyFields	CHARACTER	dataprop.i	X	X	X	–
KeyFields	CHARACTER	seleprop.i	X	X	X	X

Table 13: Public Properties By Name: L

Property Name	Data Type	Declared in	X	G	S	I
Label[1]	CHARACTER	seleprop.i	X	X	X	X
LabelHandle	HANDLE	seleprop.i	X	–	–	–
LabelHandles	CHARACTER	filtprop.i	X	–	–	–
LastDBRowIdent	CHARACTER	qryprop.i	X	–	X	–
LastResultRow	CHARACTER	dataprop.i	X	X	X	–
LastRowNum	INTEGER	dataprop.i	X	X	X	–
LayoutOptions	CHARACTER	visprop.i	X	X	X	–
LayoutVariable	CHARACTER	visprop.i	X	X	-	–
ListInitialized	LOGICAL	seleprop.i	X	–	–	–
LoadedByRouter	LOGICAL	b2bprop.i	X	X	X	–
LogFile	CHARACTER	consprop.i	X	X	X	X
LogicalValues	CHARACTER	filtprop.i	-	-	-	–

[1] The label string if defined locally, "?" if defined in the data source, or the empty string if not defined.

Table 14: Public Properties By Name: M *(1 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
MapNameProducer	CHARACTER	b2bprop.i	X	X	X	X
MapObjectProducer	CHARACTER	b2bprop.i	X	X	X	X
MapTypeProducer	CHARACTER	b2bprop.i	X	X	X	X
MarginPixels	INTEGER	panlprop.i	X	X	–	–

Table 14: Public Properties By Name: M *(2 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
MasterDataObject[1]	HANDLE	sboprop.i	X	X	–	–
MaxWidth	DECIMAL	filtprop.i	X	–	–	X
MaxWidth	DECIMAL	brsprop.i	X	X	X	X
Menu[2]	LOGICAL	toolprop.i	X	X	X	X
MenubarHandle	HANDLE	toolprop.i	X	–	–	–
MessageType	CHARACTER	messprop.i	X	X	X	X
ModifiedFields	CHARACTER	brsprop.i	X	–	–	–
Modify	LOGICAL	seleprop.i	X	–	–	–

1 The master data object is the SDO whose changes control behavior in other SDOs.

2 TRUE if the menu component is active.

Table 15: Public Properties By Name: N *(1 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
NameList	CHARACTER	b2bprop.i	X	X	X	X
NameSpaceHandle	HANDLE	b2bprop.i	X	X	–	–
NavigationSource	CHARACTER	qryprop.i	X	X	X	–
NavigationSource	CHARACTER	sboprop.i	X	X	X	–
NavigationSourceEvents[1]	CHARACTER	qryprop.i	X	X	–	–
NavigationSourceEvents[1]	CHARACTER	sboprop.i	X	X	–	–
NavigationTarget	CHARACTER	panlprop.i	X	X	X	–
NavigationTargetEvents[1]	CHARACTER	panlprop.i	X	X	X	–

Table 15: Public Properties By Name: N *(2 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
NavigationTargetName	CHARACTER	panlprop.i	X	X	X	X
NewRecord[2]	CHARACTER	dvisprop.i	X	X	–	–
NumDown	INTEGER	brsprop.i	X	X	X	X
NumRows	INTEGER	seleprop.i	X	X	X	X

1 Comma-separated list of the events to which this object subscribes in the respective source/target.

2 Possible values: "Add", "Copy", and "No".

Table 16: Public Properties By Name: O *(1 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
ObjectEnabled	LOGICAL	visprop.i	X	X	–	–
ObjectHidden[1]	LOGICAL	smrtprop.i	X	X	–	–
ObjectInitialized	LOGICAL	smrtprop.i	X	X	–	–
ObjectLayout	CHARACTER	visprop.i	X	X	X	X
ObjectMapping[2]	CHARACTER	sboprop.i	X	X	X	–
ObjectName	CHARACTER	smrtprop.i	X	X	X	X
ObjectPage	INTEGER	smrtprop.i	X	X	–	–
ObjectType	CHARACTER	smrtprop.i	X	X	–	–
ObjectVersion[3]	CHARACTER	smrtprop.i	X	X	–	–
OpenOnInit	LOGICAL	sboprop.i	X	x	x	-
OpenOnInit	LOGICAL	qryprop.i	X	X	-	–
OpenQuery[4]	CHARACTER	qryprop.i	-	X	X	–
Operator	CHARACTER	filtprop.i	X	X	X	X
OperatorHandles	CHARACTER	filtprop.i	X	–	–	–
OperatorLongValues	CHARACTER	filtprop.i	X	X	–	–
OperatorShortValues	CHARACTER	filtprop.i	X	–	–	–
OperatorStyle	CHARACTER	filtprop.i	X	X	X	X
OperatorViewAs	CHARACTER	filtprop.i	X	X	X	X
Optional	LOGICAL	seleprop.i	X	X	X	X
OptionalBlank[5]	CHARACTER	seleprop.i	X	X	X	–

Table 16: Public Properties By Name: O *(2 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
OptionalString	CHARACTER	seleprop.i	X	X	X	X
OutMessageSource	HANDLE	prodprop.i	X	X	X	–
OutMessageTarget	HANDLE	msghprop.i	X	X	X	–
OutMessageTarget	HANDLE	cntnprop.i	X	X	X	–

1 The state HIDDEN is a logical rather than visual state; it can be applied to a nonvisible object to indicate it is, for example, inactive.

2 Intended for internal-use only. Comma-separated list of handle pairs in the form sourceA,targetA,sourceB,targetB,.... The source is an SDO inside the SBO; the target is the object that serves as its DATA-TARGET and, possibly, its UPDATE-SOURCE.

3 A string representing the ADM version. For Version 9 and later objects, the string will be "ADM2.0" or later.

4 Original WHERE clause set at design time, set as the value of the OPEN-QUERY macro. By using this value as the argument to Query-Prepare, run-time modifications made under program control can be discarded.

5 TRUE if the character field evaluates to BLANK.

Table 17: Public Properties By Name: P *(1 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
PageNTarget[1]	CHARACTER	cntnprop.i	X	X	X	–
PageSource	HANDLE	cntnprop.i	X	X	X	–
PageTarget	CHARACTER	folder.w	X	X	X	–
PageTargetEvents	CHARACTER	folder.w	X	X	–	–
PanelFrame	HANDLE	panlprop.i	X	X	–	–
PanelLabel	HANDLE	panlprop.i	X	X	–	–
PanelState	CHARACTER	panlprop.i	X	X	X	–
PanelType[2]	CHARACTER	panlprop.i	X	X	X	X

Table 17: Public Properties By Name: P *(2 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
Persistency	CHARACTER	prodprop.i	X	X	X	X
PingInterval	INTEGER	messprop.i	X	X	X	X
Priority	INTEGER	prodprop.i	X	X	X	X
PromptLogin	LOGICAL	messprop.i	X	X	X	X
PropertyDialog[3]	CHARACTER	smrtprop.i	X	X	–	–

[1] The list of all objects not assigned to Page 0. Do not manipulate this property by hand, as it has a special format. Call addLink instead.

[2] Valid values: "UPDATE", "NAVIGATION".

[3] Identifies the dialog procedure that sets instance properties. Normally only used internally.

Table 18: Public Properties By Name: Q

Property Name	Data Type	Declared in	X	G	S	I
QueryColumns	CHARACTER	qryprop.i	X	–	–	–
QueryContainer	LOGICAL	dataprop.i	X	X	–	–
QueryContext	CHARACTER	dataprop.i	X	X	X	-
QueryHandle	HANDLE	qryprop.i	X	X	–	–
QueryObject[1]	LOGICAL	smrtprop.i	-	X	–	–
QueryPosition[2]	CHARACTER	qryprop.i	-	X	X	–
QueryRowIdent	CHARACTER	qryprop.i	X	X	X	–
QueryRowObject	HANDLE	brosprop.i	X	X	X	-
QueryString	CHARACTER	qryprop.i	X	X	X	–

[1] TRUE if the object manages a local query. Note that local queries are deprecated except in objects, such as the SDO, dedicated to data access.

[2] Valid values are "FirstRecord", "LastRecord", "NotFirstOrLast", "OnlyRecord", and "NoRecordAvailable".

Table 19: Public Properties By Name: R *(1 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
RebuildOnRepos[1]	LOGICAL	dataprop.i	X	X	X	X
RecordState	CHARACTER	dvisprop.i	X	X	–	–
ReplyConsumer	HANDLE	prodprop.i	X	–	–	–
ReplyReqList	CHARACTER	b2bprop.i	X	X	X	X
ReplyRequired	LOGICAL	msghprop.i	X	X	X	–
ReplySelector	CHARACTER	msghprop.i	X	X	X	–
ReplySelectorList	CHARACTER	b2bprop.i	X	X	X	X
RepositionDataSource[2]	LOGICAL	seleprop.i	X	X	X	X
RightToLeft	CHARACTER	panlprop.i	X	X	X	X
RouterSource	CHARACTER	routprop.i	X	X	X	–
RouterTarget	HANDLE	consprop.i	X	X	X	–
RowIdent[3]	CHARACTER	dvisprop.i	-	X	X	–
RowObject[4]	HANDLE	dataprop.i	X	X	–	–
RowObjectState[5]	CHARACTER	dataprop.i	-	X	X	–
RowObjectState[5]	CHARACTER	sboprop.i	-	X	X	–
RowObjectTable	HANDLE	dataprop.i	-	X	X	–
RowObjUpd	HANDLE	dataprop.i	X	X	–	–
RowObjUpdTable[6]	HANDLE	dataprop.i	X	X	–	–

Table 19: Public Properties By Name: R *(2 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
RowsToBatch	INTEGER	dataprop.i	X	X	X	X
RunDOOptions	CHARACTER	cntnprop.i	X	X	X	–
RunMultiple	LOGICAL	cntnprop.i	X	X	X	–

[1] TRUE if the RowObject temp-table should be rebuilt whenever an attempt is made to point at a record outside the current dataset.

[2] TRUE if the data-source is to be repositioned whenever the object experiences the VALUE-CHANGED event. Not needed for the view-as-browse option. Set this to true when the data-source also serves other objects, if they need to be refreshed when this object experiences VALUE-CHANGED.

[3] RowIdent for current record in RowObject temp-table. Takes the form of a comma-delimited string composed of the ROWID of the temp-table record followed by the ROWIDs of the records from which the temp-table record was made.

[4] Handle of the RowObject buffer.

[5] TRUE if uncommitted updates present. Valid values are "NoUpdates" and "RowUpdated"

[6] Handle of buffer where updates are stored.

Table 20: Public Properties By Name: S *(1 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
SaveSource	LOGICAL	dvisprop.i	X	–	X	–
SchemaHandle	HANDLE	b2bprop.i	X	X	X	–
SchemaList	CHARACTER	b2bprop.i	X	X	X	X
ScrollRemote	LOGICAL	brsprop.i	X	X	X	X
SearchField	CHARACTER	brsprop.i	X	X	X	X
SearchHandle	HANDLE	brsprop.i	X	–	–	–
SelectionHandle	HANDLE	seleprop.i	X	–	–	–
SelectionImage	CHARACTER	seleprop.i	X	–	–	–
Selectors	CHARACTER	consprop.i	X	X	X	X

Table 20: Public Properties By Name: S *(2 of 2)*

Property Name	Data Type	Declared in	X	G	S	I
ServerFileName	CHARACTER	appsprop.i	–	X	X	–
ServerOperatingMode	CHARACTER	appstaprop.i	X	X	X	X
ServerSubmitValidation	LOGICAL	dataprop.i	–	X	X	–
ShowBorder[1]	LOGICAL	toolprop.i	X	X	X	X
ShutDownDest	CHARACTER	consprop.i	X	X	X	X
Sort	LOGICAL	seleprop.i	X	X	X	X
StartBrowseKeys[2]	CHARACTER	seleprop.i	X	X	X	X
Starting	LOGICAL	seleprop.i	X	–	–	–
StartPage	INTEGER	cntnprop.i	X	–	–	–
StatelessSavedProperties	CHARACTER	dataprop.i	X	–	–	–
StopConsumer	HANDLE	consprop.i	X	–	–	–
SubModules	CHARACTER	toolprop.i	X	X	X	X
Subscriptions	CHARACTER	consprop.i	X	X	X	X
SupportedLinks	CHARACTER	smrtprop.i	X	X	X	X
SupportedMessageTypes	CHARACTER	messprop.i	X	X	–	–

1 TRUE if a 3D border should be displayed around buttons and as RULE delimiters.

2 List of KeyLabels/KeyFunctions that start the browse if DefineAnyKeyTrigger is TRUE.

Table 21: Public Properties By Name: T

Property Name	Data Type	Declared in	X	G	S	I
TableIOSource	HANDLE	dvisprop.i	X	X	X	-
TableIOSourceEvents[1]	CHARACTER	dvisprop.i	X	X	X	-
TableIOTarget[2]	CHARACTER	panlprop.i	X	X	X	-
TableIOTargetEvents[1]	CHARACTER	panlprop.i	X	X	X	-
TableIOType[3]	CHARACTER	toolprop.i	X	X	X	X
Tables[4]	CHARACTER	qryprop.i	-	X	X	-
TimeToLive	DECIMAL	prodprop.i	X	X	X	X
Toolbar[5]	LOGICAL	toolprop.i	X	X	X	X
ToolbarMinWidth	DECIMAL	toolprop.i	X	-	-	-
ToolHeightPxl	INTEGER	toolprop.i	X	-	-	-
ToolMarginPxl	INTEGER	toolprop.i	X	-	-	-
ToolSeparatorPxl	INTEGER	toolprop.i	X	-	-	-
ToolSpacingPxl	INTEGER	toolprop.i	X	-	-	-
ToolTip	CHARACTER	seleprop.i	X	X	X	X
ToolWidthPxl	INTEGER	toolprop.i	X	-	-	-
TranslatableProperties	CHARACTER	smrtprop.i	X	X	X	-
TypeName	CHARACTER	b2bprop.i	X	X	X	-

1 Comma-delimited list of events in the respective source/target to which this object subscribes.

2 List of handles to this object's TableIO-Targets.

3 List of handles to the object's TableIO-Targets.

[4] List of database tables in the QueryObject. Get function defined in both `query.p` and `data.p`.

[5] TRUE if toolbar is to be created.

Table 22: Public Properties By Name: U

Property Name	Data Type	Declared in	X	G	S	I
UIBMode[1]	CHARACTER	smrtprop.i	-	X	X	-
UpdatableColumnsByTable	CHARACTER	qryprop.i	X	-	-	-
UpdateSource	CHARACTER	cntnprop.i	X	X	X	-
UpdateSource	CHARACTER	cntnprop.i	X	X	X	-
UpdateSource	HANDLE	dataprop.i	X	X	X	-
UpdateStateInProcess	LOGICAL	sboprop.i	X	-	-	-
UpdateTarget	CHARACTER	cntnprop.i	X	X	X	-
UpdateTarget	CHARACTER	dvisprop.i	X	X	X	-
UpdateTargetNames	CHARACTER	dvisprop.i	-	X	X	X
UpdatingRecord	LOGICAL	panlprop.i	X	-	X	-
UseBegins	LOGICAL	filtprop.i	X	X	X	X
UseContains	LOGICAL	filtprop.i	X	X	X	X

[1] TRUE if application is under control of AppBuilder (UIB—User-Interface Builder—was an earlier name for AppBuilder). Valid values are "Design", "Design-Child", and the empty string.

Table 23: Public Properties By Name: V

Property Name	Data Type	Declared in	X	G	S	I
ValidateOnLoad	LOGICAL	xmlprop.i	X	X	X	-
ViewAs[1]	CHARACTER	seleprop.i	X	X	X	X
ViewAsFields	CHARACTER	filtprop.i	X	X	X	X
VisibleRowids	CHARACTER	brsprop.i	X	X	X	-
VisibleRowReset	LOGICAL	brsprop.i	X	X	X	-
VisualBlank	CHARACTER	filtprop.i	X	X	X	–

[1] String defining how the SmartSelect is to display. Valid values are "Browse", "Selection-list", "Radio-set:H", "Radio-set:V", "Combo-box:Drop-down-List" (the default), and "Combo-box:Simple".

Table 24: Public Properties By Name: W

Property Name	Data Type	Declared in	X	G	S	I
WaitForObject	HANDLE	cntnprop.i	X	X	X	–
Waiting	LOGICAL	consprop.i	X	X	X	–

Alphabetical Listing of API Routines

This listing includes all routines except those specific to the WebSpeed environment (see the "Alphabetical Listing Of WebSpeed-specific API Routines" chapter).

The filename in parentheses that follows the identifier indicates where you can find the source for the routine.

actionAccelerator (action.p)

Returns the accelerator string, if any, for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: CHARACTER

Notes: None

actionAccessType (action.p)

Returns the access type, if any, for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: CHARACTER

Notes: None

actionCanRun (toolbar.p)

Returns TRUE if the target is valid and the procedure exists in it.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest, of type "RUN".

Returns: LOGICAL

Notes: Called from buildMenu and createToolbar.

actionCaption (action.p)

Returns the caption for some action, or the empty string if there is no caption defined.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: CHARACTER

Notes: None

actionCaption (toolbar.p)

Override action class and caption.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The replacement action

Returns: CHARACTER

Notes: None.

actionCategory (action.p)

Returns the category value for the action identified in the argument.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest

Returns: CHARACTER

Notes: None.

actionCategoryIsHidden (toolbar.p)

Returns TRUE if the action’s category is hidden

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action to be evaluated

Returns: LOGICAL

Notes: None.

actionChecked (toolbar.p)

Returns TRUE if its call to get*propname* returns a value that matches the actions checked.

Parameters:

```
INPUT pcAction AS CHARACTER
```

Action ID of an action of type "RUN".

Returns: LOGICAL

Notes: Currently we only support logical values. If the get function does not exist, or the object is not valid, this function returns the undefined value.

actionChildren (action.p)

Returns a comma-separated list of all children of some action.

Parameters:

```
INPUT pcId AS CHARACTER
```

The parent action id.

Returns: CHARACTER

Notes: Class actions are shared by all instances, so class actions are meant to be returned first for the sake of consistency. However, if you define instance actions before class actions, they have a lower order and are returned first in the first realization of the object. This discrepancy is automatically corrected as the object is realized again and again. You can manipulate the order using insertMenu() and createToolbar (in toolbar.p).

actionControlType (action.p)

Returns the control-type value for the action identified in the argument.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest

Returns: CHARACTER

Notes: None.

actionCreateEvent (action.p)

Returns the createEvent for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: CHARACTER

Notes: Cannot be changed.

actionDescription (action.p)

Returns the description value for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: CHARACTER

Notes: None

actionDisabled (action.p)

Returns TRUE if the action identified in the argument has been disabled.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest

Returns: LOGICAL

Notes: None.

actionEnableRule (action.p)

Returns the EnableRule value for the action passed as the argument.

Parameters:

```
INPUT pcAction AS CHARACTER
```

Returns: CHARACTER

Notes: None.

actionEnableRulex (action.p)

Assembles and returns some query string, depending on the argument.

Parameters:

```
INPUT pcAction AS CHARACTER
```

Valid values are ‘reset’, ‘add’, ‘copy’, ‘delete’, ‘save’, ‘cancel’, ‘first’, ‘prev’, ‘next’, and ‘last’.

Returns: CHARACTER

Notes: None.

actionGroups (action.p)

Returns a comma-separated lists of actions, grouped.

Parameters: None

Returns: CHARACTER

Notes: None

actionHideRule (action.p)

Returns the value of HideRule for the action supplied as the argument.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest

Returns: CHARACTER

Notes: None.

actionHideRuleX (action.p)

Returns some string depending on the argument. The empty string is returned if the argument is not valid.

Parameters:

```
INPUT pcAction AS CHARACTER
```

Valid values are ‘update’, ‘txtTableioOk’, and ‘txtTableioCancel’

Returns: CHARACTER

Notes: None.

actionImage (action.p)

Returns the Image value for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: CHARACTER

Notes: None

actionImageAlternate (action.p)

Returns the value for Image2, for the action identified by the argument.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest

Returns: CHARACTER

Notes: None.

actionImageAlternateRule (action.p)

Returns the value for ImageAlternateRule, for the action identified by the argument.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest

Returns: CHARACTER

Notes: None.

actionInitCode (action.p)

Returns the string value of the InitCode column for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: CHARACTER

Notes: None

actionIsMenu (action.p)

Returns TRUE if the action is a menu. Actions defined as a menu are considered to be a constant part of the toolbar and are not selectable. This means that the action is always available. The action needs to be added to a toolbar with createToolbar or insertMenu(). It will **not** appear as a selectable action in the instance property dialog even if actionIsParent returns a TRUE status.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: LOGICAL

Notes: None

actionIsParent (action.p)

Returns TRUE if the identified action is a parent.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: LOGICAL

Notes: Traverses the actions table testing whether any action has this action as its parent.

actionLabel (action.p)

Returns the label on the action’s button

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action whose button label is wanted

Returns: CHARACTER

Notes: actionLabel is an alias for actionName

actionLabel (toolbar.p)

Overrides an action’s label.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The overriding action

Returns: CHARACTER

Notes: None.

actionLink (action.p)

Returns the Link value for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: LOGICAL

Notes: None

actionLogicalObjectName (action.p)

Returns the LogicalObjectName value for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: CHARACTER

Notes: None.

actionName (action.p)

Returns the Name value for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: LOGICAL

Notes: None

actionOnChoose (action.p)

Returns the OnChoose value for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: LOGICAL

Notes: None

actionParameter (action.p)

Returns the RunParameter value for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: CHARACTER

Notes: None.

actionParent (action.p)

Returns the Parent value for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: LOGICAL

Notes: None

actionPhysicalObjectName (action.p)

Returns the PhysicalObjectName value for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest

Returns: CHARACTER

Notes: None.

actionPublishCreate (toolbar.p)

Subscribes create events to objects.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: LOGICAL

Notes:

- Subscribes both active and inactive/hidden objects. Target links are considered as multiple, source links as single.
- Using the publish/subscribe mechanism makes it possible to refer to SOURCE-PROCEDURE in the events.

actionRefresh (action.p)

Returns TRUE if some action can be refreshed.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: LOGICAL

Notes: None

actionRunAttribute (action.p)

Returns the RunAttribute value for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest

Returns: CHARACTER

Notes: None.

actionSecondImage (action.p)

Returns the Image2 value for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: CHARACTER

Notes: None.

actionSecuredToken (action.p)

Returns the SecurityToken value for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest

Returns: CHARACTER

Notes: None.

actionSubstituteProperty (action.p)

Returns the SubstituteProperty value for some action

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest

Returns: CHARACTER

Notes: None.

actionTarget (toolbar.p)

Returns the handle of the target, by default the handle of CONTAINER-SOURCE.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest.

Returns: HANDLE

Notes: Called by actions of type "RUN" and "PROPERTY".

actionTooltip (action.p)

Returns some other value depending on the value of TranslatedActionTooltip for some action. The values that might be returned are: TranslatedActionMLabel, TranslatedActionLabel, or Tooltip.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest

Returns: CHARACTER

Notes: None.

actionTooltip (toolbar.p)

Overrides a action’s tooltip.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The overriding action

Returns: CHARACTER

Notes: None.

actionType (action.p)

Returns the Type value for some action.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest

Returns: LOGICAL

Notes: None

activeTarget (panel.p)

Return the targets of some link type.

Parameters:

```
INPUT pcLink AS CHARACTER
```

The type of link: “TableIO”, “Navigation”, “Commit”

Returns: HANDLE

Notes:

- The toolbar only supports one active object in these, but it might be linked to inactive objects on hidden pages. If more than one target this procedure returns the active hidden object where ObjectHidden = FALSE for some link type.
- This is not overridden by Toolbar.

addDataTarget (sbo.p)

Updates the ObjectMapping property. This property is used to broker messages between the contained objects and outside objects in communication with them. It is also used to set DataSourceNames and UpdateTargetNames in the data-targets.

Parameters:

```
INPUT phTarget AS HANDLE
```

The target handle to be added.

Returns: Nothing (procedure)

Notes:

- Called by registerObject, subscribed as DataTargetEvent and published from the DataTarget's initializeObject.
- DataSourceNames might be specified by the user in which case it actually specifies how to generate the ObjectMapping. If it is not set, both it and UpdateTargetNames are **always** set here so that colValues, addRow, deleteRow, updateRow, etc. can identify the intended target or source without looping through all the fields again and again.
- Objects built against RowObject must find ALL columns in ONE of the ContainedDataObjects in order to become mapped.
- Only this procedure is allowed to add Data-Targets to the ObjectMapping property.
- ObjectMapping versus DataSourceNames. There is some overlap here and add-, copy- and deleteRow with ObjectMapping could be used instead of DataSourceNames. But since both cases require knowing the requester, there is not much advantage to only using the ObjectMapping. A way to distinguish between UpdateTargets and DataSources is necessary, and having them implemented similarly makes it all a bit easier to use.

  Ideally, the SBO should not have to know about or deal with the object mapping. That should happen at the visual level instead, so the way mapping works might change in some future release.

addLink (smart.p)

Adds a link between two objects by setting property values in each.

Parameters:

`INPUT phSource AS HANDLE`

Source procedure handle.

`INPUT pcLink AS CHARACTER`

Link name.

`INPUT phTarget AS HANDLE`

Target procedure handle.

Returns: Nothing (procedure)

Notes:

- Normally addLink is run from code generated by the AppBuilder in the internal procedure adm–create–objects, in response to the developer adding links to a SmartContainer at design time. Additional calls to addLink can be written into an application when additional links are needed at run time.
- If the link is not in the SupportedLinks list for either object, then the link name will be treated as a single subscription in the “Target” for an event of that name in the “Source”. See the first example below.
- If the link is in the list of PassThroughLinks, and the object at one end or the other of the new link is a SmartContainer, addLink checks to see if a link of the same type exists for that container. If so, the two links are combined, or chained together, into a single link that connects the original Source with the final Target, bypassing the Container. Refer to documentation for ADM@ or Progress AppBuilder for more information about PassThrough links.
- If the link name is of the type “Page*N*", where *N* is an integer, then the caller is defining the Target to be on logical page *n* of the Source, which must be a SmartContainer. The addLink procedure will add the Target to the special link name *PageNTargets* in the Source.

- For each entry in an object's SupportedLinks, there must be a property that stores the handle(s) of the object(s) at the other end of the link, and functions to set and get that property. There must also be a property that stores a list of the named events which are associated with that link. For example, if Navigation–Source is one of a SmartPanel's SupportedLinks, and Navigation–Target is one of a SmartDataObject's SupportedLinks, in executing the second example above, addLink adds hdCust to the NavigationTarget property of the SmartPanel, and hSmartPanel to the NavigationSource property of the SmartDataObject. The addLink procedure checks the RETURN data type of the "get" property functions for these properties: a RETURN type of HANDLE means that only a single object is supported on that end of the link and that the property value is stored as a value of type HANDLE. If the RETURN type is CHARACTER, multiple objects are supported on that end of the link and the object handle is added to a property value stored as a comma-separated list of object handles. For example, the NavigationTarget property is CHARACTER, because a panel may have multiple Navigation–Targets. The NavigationSource property is HANDLE because an SmartDataObject may have only one Navigation–Source. The NavigationSourceEvents property for the SmartDataObject stores a list of the events to be subscribed to in the Source (see second example below).

Examples:

```
/*  This example will define a dynamic link from MyWindow to the
    SDOdCust. If the link type SpecialEvent is not defined as a SupportedLink
    for either SmartWindows or SmartDataObjects, then addLink will do a
    single SUBSCRIBE in hdCust to 'SpecialEvent' in hMyWindow. This means
    that when code in MyWindow does 'PUBLISH 'SpecialEvent'." The
    user-defined internal procedure named SpecialEvent in dCust will be
    executed.*/

RUN addLink (INPUT hMyWindow, INPUT 'SpecialEvent':U, INPUT hdCust).

/*  This example will add an instance of the SupportedLink 'Navigation'
    from a SmartPanel to an SDO. Normally this link would be defined at
    design time and the call to addLink generated automatically by the
    if the SmartPanel were run after startup by application code, if
    response to some application event, then the link would need to be
    created by application code also. AddLink will SUBSCRIBE the SDO to
    fetchFirst, fetchNext, fetchPrev, and fetchLast in the SmartPanel,
    because these events are listed in the SmartPanel property
    NavigationSourceEvents in the SDO. */

RUN addLink (INPUT hSmartPanel, INPUT 'Navigation':U, INPUT hdCust).
```

addMessage (smart.p)

Inserts the message text into a data message log along with its Field, and Table if known.

Parameters:

`INPUT pcText AS CHARACTER`

Text of the message.

`INPUT pcField AS CHARACTER`

The field name for which the message occurred, if it was related to a specific field.

`INPUT pcTable AS CHARACTER`

The database table for which the message occurred, if it was related to an update to a database table.

Returns: Nothing (procedure)

Notes:

- If pcText is unknown (?), that signals that this function should retrieve messages from the error–status system handle.
- The message log is a character string in a special format intended to be decoded with the functions fetchMessages or showDataMessages.
- Message texts that are intended to be seen by end users can be enabled for translation into other languages by putting them into the application code as quoted strings, and then using the Translation Manager tool. Generally, ADM messages which are expected to be seen by developers, for example, messages that indicate errors in the application design, are entered into the ADM super procedures as quoted strings with the :U suffix so that they will not be seen by the Translation Manager tool.

- If the unknown value (?) is passed as the message text value, then addMessage will retrieve as many error messages as are stored in the ERROR–STATUS handle using the GET–MESSAGE method. This would be appropriate, for example, after executing a database access statement NO–ERROR, then checking the value of ERROR–STATUS:ERROR.

Examples:

```
/*  This example adds a specific message to the log for the SDO field
    whose name is stored in the variable cField, for no particular
    database table. */

RUN addMessage("This operation could not be completed.":U, cField, ?).
```

addNavigationSource (sbo.p)

Adds a NavigationSource to the ObjectMapping property that is used to broker messages between contained objects and the outside objects with which they communicate.

Parameters:

```
INPUT phSource AS HANDLE
```

The handle of the source.

Returns: Nothing (procedure)

Notes: Called by registerObject, which is subscribed as NavigationSourceEvent and published by the Navigation-Source's initializeObject.

addNode (treeview.p)

Adds a node to the tree.

Parameters:

```
INPUT phBuffer AS HANDLE
```

Handle to buffer of temp-table tTreeData, used to defined the node. The temp-table includes the following fields: (as defined in TVController.i)

node_key	Unique key of node
parent_node_key	Key of either parent node or sibling node (depending on node_insert)
node_label	Label of node that appears in treeview
private_data	Information stored for the node in the node's TAG property
image	Relative path and filename of the image of the node
selected_image	Relative path and filename of the node when it is selected. If blank, it uses the same image as the image field.
node_insert	Specifies where to insert the node, relative to the parent: 0 - As first node at same level as the parent_node_key 1 - As last node at same level as the parent_node_key 2 - After (next) the parent_node_key 3 - Before (previous) the parent_node_key 4 - As child of parent_node_key
node_sort	If TRUE, sort node (This must be specified for all nodes within the same level). Default is FALSE.
node_expanded	If TRUE, expand node upon adding it to TreeView. Default is FALSE.
node_checked	If TRUE, and property 'ShowCheckBoxes' is TRUE, the node appears checked. Default is FALSE

Returns: Nothing (procedure).

Notes: This procedure is called from populateTree, but can be called by itself.

addQueryWhere (query.p)

Adds some string-expression to the query's WHERE clause and stores the result in the QueryString property. It returns TRUE if successful, FALSE if an appropriate buffer name for the WHERE-clause cannot be located.

Parameters:

`INPUT pcWhere AS CHARACTER`

Expression to add (might also be an "OF" phrase).

`INPUT pcBuffer AS CHARACTER`

An optional buffer specification.

`INPUT pcAndOr AS CHARACTER`

Specifies the operator that is used to add the new expression to an existing expression or expressions, either AND (the default) or OR.

Returns: LOGICAL

Notes:

- Returns FALSE if it cannot find a buffer name to associate with the WHERE clause.
- This procedure is designed to run on the client in order to be called possibly multiple times (to add multiple phrases to the where-clause) before the full where-clause is used to reopen the query, and will operate on the attribute QueryString. openQuery takes care of the preparation of the QueryString attribute.

addQueryWhere (sbo.p)

SmartBusinessObject version of this where-clause function. It simply passes the parameters on to the SmartDataObject named in the pcBuffer argument.

Parameters:

`INPUT pcWhere AS CHARACTER`

Same as in query.p function addQueryWhere.

`INPUT pcObject AS CHARACTER`

Must match a SmartDataObject ObjectName.

`INPUT pcAndOr AS CHARACTER`

Same as in the query.p function.

Returns: LOGICAL

Notes: Currently the pcBuffer argument must be specified.

addRecord (browser.p)

SmartDataBrowser version of addRecord. Initiates the creation of a new record. Inserts a new row below the current row, if any. The ROW–ENTRY trigger in the SmartDataBrowser (%DLC%/src/adm2/brsentry.i) displays initial values.

Parameters: None

Returns: Nothing (procedure)

Notes:

- Invokes addRecord version in `datavis.p` (RUN SUPER).
- Sets the BrowseInitted property to FALSE. This property is used in the ROW–ENTRY trigger where the actual display of initial values takes place to make sure that the initialization logic for add is executed once.

- Invoked when an Add is initiated (typically by choosing the Add button in an Update SmartPanel, or the Add button or Add menu item in a SmartToolbar).
- Customized when additional processing is needed at the start of an Add operation. When an addRecord override is invoked, the new record has not yet been created, and there is no transaction active.

addRecord (datavis.p)

Initiates the creation of a new record. First verifies that there is no update pending. If there is an update pending, notifies the user that current values must be saved or cancelled before the Add operation takes place, then it displays initial values.

Parameters: None

Returns: Nothing (procedure)

Notes:

- Publishes updateState “Update” to signal related objects that the update is in progress.
- Sets the NewRecord property to "Add".
- Invokes enableFields. The addRow method in the associated SmartDataObject will be invoked at a lower level (from addRecord in `viewer.p` for a SmartDataViewer or in browse triggers for a SmartDataBrowser).
- Is invoked when an Add is initiated (typically by choosing the Add button in an Update SmartPanel, or the Add button or Add menu item in a SmartToolbar).

- Is customized when additional processing is needed in the visual SmartObject at the start of an Add operation. When an addRecord override is invoked, the new record has not yet been created, and there is no transaction active. If you want to customize the Add operation for the SmartDataObject that manages the table being shown in the visualization, you can override addRow to customize the processing at the time the new row is added to the RowObject temp–table, or override submitRow to customize the saving of the newly added row.

Examples:

```
PROCEDURE addRecord:

/*
** Code placed here will execute PRIOR to standard behavior.
*/
  RUN SUPER.
/*
** Code placed here will execute AFTER standard behavior.
*/
  iRecTotal = iRecTotal + 1.
  DISPLAY iRecTotal WITH FRAME {&FRAME-NAME}.

END PROCEDURE.
```

addRecord (viewer.p)

Initiates the creation of a new record in the RowObject temp-table. Initial values for the new record are displayed in the viewer's frame. Keeps a running total of the records added.

Parameters: None

Returns: Nothing (procedure).

Notes:

- addRow() is invoked in the update target, which creates the new RowObject temp-table record and returns the column values. When the SmartDataViewer is a GroupAssign target, its GroupAssign source has already added the record, therefore only the values of the columns are retrieved.
- Column values are passed to displayFields, which displays the values in the viewer's frame.
- The record is not actually created in the database until it is committed. This is done in assignDBRow procedure.

- For Add and Copy, it is important that key fields, and any other fields assigned by the CREATE trigger, not be set to updateable in the SmartDataObject and enabled for data entry. If they are, the screen values are assigned on top of the key values set by the trigger, with an error as the likely result.
- applyEntry is run to apply entry to the first field in the frame after the initial column values are displayed.
- The procedure submitForeignKey in `data.p` automatically assigns key values inherited as ForeignField values from another SmartDataObject. For example, in an Order SmartDataViewer used for adding new Orders for the current Customer selected in another object, the CustNum field can (and normally should) be left disabled in the Order SmartDataViewer. Its value is assigned in the Order SmartDataObject to the CustNum value from the parent Customer SmartDataObject at the time the new record is Saved. You can override submitForeignKey in the SmartDataObject your Data visualization object is connected to in order to write any additional special logic you want to apply when adding or copying a record.
- addRecord is invoked when an Add is initiated, typically by choosing the Add button in an Update SmartPanel, or the Add button or Add menu item in a SmartToolbar.
- addRecord republishes the addRecord event to handle the Add for GroupAssign targets.
- addRecord can be customized when additional processing is needed at the start of an Add operation. Keep in mind that when an addRecord override is invoked, the new record has not yet been created, and there is no transaction active.

addRow (data.p)

Creates a new RowObject temp-table record, initializes it, and returns a list of ColumnName/Value pairs to be displayed by the SmartDataViewer that called this procedure.

Parameters:

```
INPUT pcViewColList AS CHARACTER
```

A comma-separated list of column names that are to be displayed in the SmartDataViewer that called addRow.

Returns: CHARACTER

Notes: None

addRow (sbo.p)

SmartBusinessObject version of the function. It passes the column list on to the contained SmartDataObject that manages that data.

Parameters:

```
INPUT pcViewColList AS CHARACTER
```

Returns: CHARACTER

Notes: None

adjustActions (toolbar.p)

Called after realization of toolbar to adjust sizes

Parameters: None.

Returns: LOGICAL

Notes: Returns TRUE if change succeeded. Currently changes width of all actions in a horizontal toolbar to maxwidth encountered during realization.

adjustTabOrder (smart.p)

Changes the tab order of SmartObjects.

Parameters:

```
INPUT phObject AS HANDLE
```

Handle of the smart object.

```
INPUT phAnchor AS HANDLE
```

Handle of either another SmartObject procedure or a widget-handle of the object that will anchor the SmartObject.

```
INPUT pcPosition AS CHARACTER
```

"After" if the SmartObject is moved after the anchor.

"Before" if the SmartObject is moved before the anchor.

Returns: Nothing (procedure).

Notes: adjustTabOrder calls are generated by the AppBuilder in adm–create–objects. Calls to this procedure can be added to an application to do dynamic re-ordering of the tab order of SmartObjects in a SmartContainer at run time.

Examples:

```
/*  This example will make sure that the tab order position of a
    SmartPanel in a SmartWindow is immediately before the fill-in field
    FIELD-1 in that window. */

RUN adjustTabOrder (INPUT hSmartPanel, INPUT FIELD-1:HANDLE
   IN FRAME {&FRAME-NAME}, INPUT "BEFORE":U).
```

anyKey (combo.p)

Stub routine that can be used to trap a keypress and scroll to the first entry beginning with that key character. Add your code to complete.

Parameters: None

Returns: Nothing (procedure)

Notes: Use LAST-EVENT:FUNCTION for testing keypress.

anyKey (lookup.p)

Routine that can be used to trap a keypress and display a single-value return by going directly to the query.

Parameters: None

Returns: Nothing (procedure)

Notes: Use LAST-EVENT:FUNCTION for testing keypress.

anyKey (select.p)

Runs persistently on any key of the selection fill-in.

Parameters: None

Returns: Nothing (procedure)

Notes: Used only with the view-as browse option. The persistent trigger that calls this is defined if DefineAnyKeyTrigger is true. The user could define an override of this in selectcustom.p if more sophisticated interactions with the data-source are required without starting the browse. For example:

```
listif last-event:key-function = "cursor-down"
then run fetchNext in getDataSource().
```

anyMessage (smart.p)

Returns a flag indicating whether there is any message in the error message log.

Parameters: None

Returns: LOGICAL

Notes: Error messages generated during the execution of SmartObjects (especially SmartDataObjects) are normally saved in a message log (actually just a specially formatted character string). This assures that multiple messages can be accumulated for multiple errors that occur during an update, and that they will be returned properly to the client when the errors occur in a separate Progress session on an AppServer. If using a SmartDataObject as the data-source, check BOTH in the SmartDataObject and internally, because WebSpeed-specific errors are always stored internally.

Examples:

```
IF anyMessage() THEN
  /* code for unsuccessful update */
ELSE
  /* code for successful update */
```

appendContainedObjects (sbo.p)

Builds the list of ContainedObjects in top-down Data-link order.

Parameters:

```
INPUT-OUTPUT pcObjects AS CHARACTER

INPUT phObject AS HANDLE
```

Returns: LOGICAL PRIVATE

Notes: Private function

applyCellEntry (browser.p)

Applies "ENTRY" to the first enabled column in the browse, or to the column passed as an input parameter.

Parameters:

```
INPUT pcCellName AS CHARACTER
```

Either the name of the browse column on which to put focus or a question mark (?) to indicate the first column.

Returns: Nothing (procedure)

Notes:

- Used internally to reposition to the correct column, for example, when a validation fails for that column.
- Invoked from applyEntry when the SmartDataBrowser has enabled fields.

applyEntry (browser.p)

SmartDataBrowser version of applyEntry. Applies "ENTRY" to the first enabled column in the browse if columns are enabled, or to the first enabled object in the SmartDataBrowser.

Parameters:

```
INPUT pcField AS CHARACTER
```

Either the name of the browse column on which to put focus or the name of an object part of the SmartDataBrowser.

Returns: Nothing (procedure)

Notes:

- Invoked when the ADM code needs to give focus to a SmartDataBrowser. The applyEntry procedure runs applyCellEntry to do browse-specific repositioning to the proper browse cell.
- Customized when focus is given to a different column or object in the SmartDataBrowser or when special processing is needed. Be aware that repositioning to a specific column after a validation or update fails might alter the behavior of the SmartDataBrowser.

applyEntry (smart.p)

Applies “ENTRY” to the first enabled and visible object in the default frame (unless pcField is specified) or in the first child that is a Frame.

Parameters:

```
INPUT pcField AS CHARACTER
```

An optional field name; if specified (that is, if this parameter is not blank or unknown), the frame field of that name is positioned to.

Returns: Nothing (procedure)

Notes: None

Examples:

```
/* This example will move focus to the first enabled and visible field in
   the SmartDataViewer vCust. The handle hvCust would normally be
   available in the SmartDataViewer's container. This statement could
   follow a message indicating that values needed to be entered for the
   current record, for example.*/

RUN applyEntry IN hvCust (?).

/* This example applied focus to a specific field ("City") in a
   SmartDataViewer. This could result from field validation failing for
   field, for example. See updateRecord in the super procedure
   datavis.p for an actual example of how the ADM repositions focus to a
   specific field.*/

RUN applyEntry In hvCust ("City":U).
```

applyFilter (filter.p)

Applies filter criteria to the filter-target.

Parameters: None

Returns: Nothing (procedure)

Notes:

- The procedure parses the dynamic filter fields and builds a list of fields, operators and values to pass to the query. The fields are only added to the list if the field or the operator is MODIFIED. (A better mechanism to identify values that need to be added to the query might become necessary). If columnQuerySelection returns any criteria for the field with an operator that is not in the list to add, it will be added to a list of fields and operators to remove from the query.
- The operator for the field is:
  - In a separate widget (combo-box or radio-set).
  - If wild card is in the value use MATCHES or BEGINS.
  - GE and LE if there is a range field with a second value.
  - First blank separated entry in the field if OperatorStyle is set to "inline"
  - Implicit, specified by the Operator property.

applyLayout (visual.p)

Applies the Master or an alternate layout for a SmartObject that has multiple layouts.

Parameters: None

Returns: Nothing (procedure)

Notes:

- applyLayout is invoked from initializeObject to set the correct layout during SmartObject initialization. Runs the Layout procedure to apply the Master layout if it is already current (in order to reset the user interface to its original state), and then runs the Layout procedure (determined by the LayoutVariable property that is storing the LAYOUT–VARIABLE preprocessor value) to apply the new layout.
- applyLayout can be customized when special processing is needed to change layouts.

assignActionAccelerator (action.p)

Associates an accelerator with an action.

Parameters:

`INPUT pcId AS CHARACTER`

The action.

`INPUT pcValue AS CHARACTER`

The accelerator.

Returns: LOGICAL

Notes: None

assignActionAccessType (action.p)

Associates an access type with an action.

Parameters:

`INPUT pcId AS CHARACTER`

The action.

`INPUT pcValue AS CHARACTER`

The access type.

Returns: LOGICAL

Notes: None

assignActionAlternateImageRule (action.p)

Returns TRUE if able to successfully assign some value to the AlternateImageRule for some action.

Parameters:

`INPUT pcId AS CHARACTER`

The action of interest

`INPUT pcValue AS CHARACTER`

The value to be assigned as the AlternateImageRule

Returns: LOGICAL

Notes: None.

assignActionCaption (action.p)

Associates a caption with an action.

Parameters:

`INPUT pcId AS CHARACTER`

The action.

`INPUT pcValue AS CHARACTER`

The caption.

Returns: LOGICAL

Notes: None

assignActionDescription (action.p)

Associates a descriptor with an action.

Parameters:

`INPUT pcId AS CHARACTER`

The action.

`INPUT pcValue AS CHARACTER`

The descriptor.

Returns: LOGICAL

Notes: None

assignActionDisableRule (action.p)

Returns TRUE if able to assign some value to DisableRule for some action.

Parameters:

`INPUT pcId AS CHARACTER`

The action of interest

`INPUT pcValue AS CHARACTER`

The new value to be assigned to DisableRule

Returns: LOGICAL

Notes: None.

assignActionEnableRule (action.p)

Returns TRUE if able to assign some value to EnableRule for some action

Parameters:

`INPUT pcId AS CHARACTER`

The action of interest

`INPUT pcValue AS CHARACTER`

The new value to be assigned to EnableRule

Returns: LOGICAL

Notes: None.

assignActionHideRule (action.p)

Returns TRUE if able to assign some value to HideRule for some action

Parameters:

`INPUT pcId AS CHARACTER`

The action of interest.

`INPUT pcValue AS CHARACTER`

The new value for HideRule

Returns: LOGICAL

Notes: None.

assignActionImage (action.p)

Associates an image with an action.

Parameters:

`INPUT pcId AS CHARACTER`

The action.

`INPUT pcValue AS CHARACTER`

The image.

Returns: LOGICAL

Notes: None

assignActionLabel (action.p)

A passive wrapper for the assignActionName function.

Parameters:

`INPUT pcId AS CHARACTER`

The action of interest

`INPUT pcValue AS CHARACTER`

The new value for Name

Returns: LOGICAL

Notes: actionLabel is an alias for actionName

assignActionName (action.p)

Associates a name with an action

Parameters:

`INPUT pcId AS CHARACTER`

The action.

`INPUT pcValue AS CHARACTER`

The name.

Returns: LOGICAL

Notes: None

assignActionOrder (action.p)

Associates an integer order number with an action.

Parameters:

`INPUT pcId AS CHARACTER`

The action.

`INPUT piValue AS INTEGER`

The order.

Returns: LOGICAL

Notes: None

assignActionParent (action.p)

Associates a parent with an action.

Parameters:

`INPUT pcId AS CHARACTER`

The action.

`INPUT pcValue AS CHARACTER`

The parent.

Returns: LOGICAL

Notes: None

assignActionSecondImage (action.p)

Returns TRUE if able to assign a new value to Image2 for some action

Parameters:

`INPUT pcId AS CHARACTER`

The action of interest

`INPUT pcValue AS CHARACTER`

The new value

Returns: LOGICAL

Notes: None.

assignActionTooltip (action.p)

Returns TRUE if able to assign a new value to Tooltip for some action.

Parameters:

```
INPUT pcId AS CHARACTER
```

The action of interest.

```
INPUT pcValue AS CHARACTER
```

The new value

Returns: LOGICAL

Notes: None.

assignAttribute (xml.p)

Looks for the *pdOwner* node and assigns *pcValue* to the attribute *pcName*.

Parameters:

```
INPUT pdOwner AS DECIMAL
```

```
INPUT pcName AS CHARACTER
```

```
INPUT pcValue AS CHARACTER
```

Returns: LOGICAL

Notes: None

assignColumn (action.p)

Determines whether some action exists and, if so, associates a particular column with it.

Parameters:

```
INPUT pcObject AS CHARACTER
INPUT pcId AS CHARACTER
INPUT pcColumn AS CHARACTER
INPUT pcValue AS CHARACTER
```

Returns: LOGICAL

Notes: PRIVATE function.

assignColumnColumnLabel (dataextcols.p)

Sets the ColumnLabel text of some column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Can be any of:

- The unqualified column name of the RowObject field.
- The column name qualified by "RowObject".
- A qualified database field name, if the field is mapped to the SDO.

```
INPUT pcLabel AS CHARACTER
```

The new label text.

Returns: LOGICAL

Notes: None

assignColumnFormat (dataextcols.p)

Sets the Progress format text of a specified column.

Parameters:

`INPUT pcColumn AS CHARACTER`

The name of the column that will have its FORMAT string changed.

`INPUT pcFormat AS CHARACTER`

The new format string.

Returns: LOGICAL

Notes: None

assignColumnFormat (filter.p)

Overrides the default format of a column.

Parameters:

`INPUT pcColumn AS CHARACTER`

The column in the Filter-Target.

`INPUT pcFormat AS CHARACTER`

The overridden format.

Returns: LOGICAL

Notes: The value is stored with all overridden formats in the FieldFormats property. The private insertFieldProperty function adds the field to the correct position in the internal list. The columnFormat function uses the private fieldProperty function to retrieve it again.

assignColumnHelp (dataextcols.p)

Sets the Progress Help text of a specified column.

Parameters:

`INPUT pcColumn AS CHARACTER`

The name of the column that will have its HELP text changed.

`INPUT pcHelp AS CHARACTER`

The new help text.

Returns: LOGICAL

Notes: None

assignColumnHelpId (filter.p)

Stores the HelpID for a column.

Parameters:

`INPUT pcColumn AS CHARACTER`

The column in the Filter-Target.

`INPUT piHelpId AS CHARACTER`

The HelpID.

Returns: LOGICAL

Notes: The value will be stored with all defined HelpIds in the internal FieldHelpIds property. The private insertFieldProperty function will add the field to the correct position in the internal list, and the private function fieldProperty must be used to retrieve it.

assignColumnLabel (dataextcols.p)

Sets the Progress Column Label text of a specified column.

Parameters:

`INPUT pcColumn AS CHARACTER`

The name of the column that will have its Column Label changed.

`INPUT pcLabel AS CHARACTER`

The new Column Label text.

Returns: LOGICAL

Notes: None

assignColumnLabel (filter.p)

Overrides the default label of a column.

Parameters:

`INPUT pcColumn AS CHARACTER`

The column in the Filter-Target.

`INPUT pcLabel AS CHARACTER`

The overridden label.

Returns: LOGICAL

Notes: The value is stored with all overridden labels in the internal Label property. The private functions insertFieldProperty and fieldProperty are used to store and retrieve values.

assignColumnOperatorStyle (filter.p)

Overrides the default OperatorStyle of a column.

Parameters:

`INPUT pcColumn AS CHARACTER`

The column in the Filter-Target.

`INPUT pcStyle AS CHARACTER`

The overridden OperatorStyle.

Returns: LOGICAL

Notes: The value is stored in the internal FieldOperatorStyles property with all other fields similarly overridden. The private insertFieldProperty and fieldProperty functions are used to store and retrieve it.

assignColumnPrivateData (dataextcols.p)

Sets the Progress Private-data attribute of a specified column.

Parameters:

`INPUT pcColumn AS CHARACTER`

The name of the column that will have its Private-data string changed.

`INPUT pcData AS CHARACTER`

The new Private-data string.

Returns: LOGICAL

Notes: None

assignColumnTooltip (filter.p)

Stores the tooltip for a column.

Parameters:

`INPUT pcColumn AS CHARACTER`

The column in the Filter-Target.

`INPUT pcTooltip AS CHARACTER`

The tooltip text.

Returns: LOGICAL

Notes: This value will be stored with all other defined tooltips in the FieldToolTips property. The private functions insertFieldProperty and fieldProperty are used to store and retrieve the value.

assignColumnValExp (dataextcols.p)

Sets the Progress Validation Expression of a specified column.

Parameters:

`INPUT pcColumn AS CHARACTER`

The name of the column that will have its Validation Expression changed.

`INPUT pcValExp AS CHARACTER`

Returns: LOGICAL

Notes: This will not actually change the column’s validation, which is set irrevocably at compile time. However, it can be retrieved by an object and then used to generate validation code for the column.

assignColumnValMsg (dataextcols.p)

Sets the Progress validation message text of a specified column.

Parameters:

`INPUT pcColumn AS CHARACTER`

The name of the column that will have its validation message changed.

`INPUT pcValMsg AS CHARACTER`

The new validation message text.

Returns: LOGICAL

Notes: This does not actually change the column's message, which is set irrevocably at compile time. However, it can be retrieved by an object and then used to generate a message on some sort of dynamic validation.

assignColumnWidth (filter.p)

Overrides the default width of a column.

Parameters:

`INPUT pcColumn AS CHARACTER`

The column in the Filter-Target.

`INPUT pdWidth AS DECIMAL`

The overridden width.

Returns: LOGICAL

Notes: The default width is stored in the DefaultWidth or DefaultCharWidth properties. The value is stored in the internal FieldWidths with all other overridden widths. The private functions insertFieldProperty and fieldProperty are used to store and retrieve it.

assignContainedProperties (sbo.p)

Sets properties in contained objects using the returned value of the containedProperties function.

Parameters:

```
INPUT pcPropValues AS CHARACTER
```

A CHR(3)-separated list of properties and values from the containedProperties function.

```
INPUT pcReplace AS CHARACTER
```

A comma-separated pair of replacement properties not qualified by ObjectTypes.

Returns: LOGICAL

Notes: None

assignCurrentMappedObject (sbo.p)

Identifies the current contained Data Object for purposes of Navigation or other access by the caller.

Parameters:

```
INPUT   phRequester AS HANDLE

INPUT   pcObjectName AS CHARACTER
```

Returns: LOGICAL

Notes: This function maps the caller to the specified SmartDataObject using the ObjectMapping property.

assignDBRow (query.p)

Copies modified values from the RowObject row to the database records (which for an Update have already been retrieved and locked).

If the RowMod field is “A” for Add or “C” for Copy, the procedure creates the database records first and then does the assign.

Parameters:

`INPUT phRowObjUpd AS HANDLE`

Handle of the buffer to copy from.

Returns: Nothing (procedure)

Notes: The procedure copies over only those fields whose values were actually modified. If this is a copied record, all the fields that were enabled for update are saved.

assignLinkProperty (smart.p)

Sets a property value in one or more SmartObjects at the other end of a specified link, relative to the TARGET–PROCEDURE.

Parameters:

`INPUT pcLink AS CHARACTER`

Link type.

`INPUT pcPropName AS CHARACTER`

Property name.

`INPUT pcPropValue AS CHARACTER`

Property value.

Returns: LOGICAL (TRUE if the property set-function succeeded, else false.)

Notes:

- This is the Version 9 (ADM2) Version of the set–link–attribute procedure in Version 8. Note that only one property name and value is allowed, as opposed to the attribute-list format of Version 8.
- If the property function is not there or is invalid, or if any of the set-functions fail, this function returns FALSE.

Examples:

```
/* This example code from the updateRecord procedure makes sure that when
   an update completes, and the DataModified property is turned off in
   the SmartDataViewer which initiated the update, it is also turned off
   in all other Viewers which may be linked in a GroupAssign. */

lSuccess = dynamic-function('assignLinkProperty', INPUT
   'GroupAssign-Target':U, INPUT 'DataModified':U, 'no':U).
```

assignMaxGuess (browser.p)

Adds the argument value to the Browse widget's MAX-DATA-GUESS attribute.

Parameters:

```
INPUT piMaxGuess AS INTEGER
```

Returns: Nothing (procedure)

Notes: None

assignMaxGuess (sbo.p)

Gets the assignMaxGuess event from some contained SDO and passes it on to the appropriate Data-Target

Parameters:

```
INPUT piMaxGuess AS INTEGER
```

Returns: Nothing (procedure)

Notes: None

assignNodeValue (xml.p)

Assigns the value of a node. Looks at the node type and calls assignAttribute() or createText() as appropriate.

Parameters:

```
INPUT pdNode AS DECIMAL

INPUT pcValue AS CHARACTER
```

Returns: LOGICAL

Notes: This adds the attribute in the document to its owner.

assignPageProperty (containr.p)

Sets the specified property in all objects on the CurrentPage of a SmartContainer. If some object on the page does not have the property, it is ignored without error.

Parameters:

```
INPUT pcProp AS CHARACTER
```

The property to set.

```
INPUT pcValue AS CHARACTER
```

The value to assign to that property.

Returns: Nothing (procedure)

Notes:

- This variation on assignLinkProperty is necessary because the notion of paging does not fit well with PUBLISH/SUBSCRIBE. This is because there is a single property (PageNTargets) that stores the handles of all of a SmartContainer's objects that are not on page 0 in a specially delimited list.
- The pcValue parameter is specified in CHARACTER form but can be used to assign values to noncharacter properties.

- All objects in a Container will subscribe to initializeObject, etc., but the paging performs the operation on subsets of those objects at a time. That is, the container will not publish initializeObject to objects on a page other than zero until that page is first viewed. So properties such as HideOnInit, which are set as part of initialization, must be set page-by-page.

Examples:

```
/*
** Sets the DataModified property to "no"
*/
RUN assignPageProperty ('DataModified':U, 'no':U).
```

assignQuerySelection (query.p)

Assigns selection criteria to the query and distributes the column/value pairs to the corresponding buffer's where-clause. Each buffer's expression will always be enclosed in parentheses.

Parameters:

`INPUT pcColumns AS CHARACTER`

A comma-separated list of column names.

`INPUT pcValues AS CHARACTER`

A CHR(1)-separated list of the corresponding values.

`INPUT pcOperators AS CHARACTER`

The operator (one for all columns):

- A blank defaults to (EQ).
- A slash defines alternative string operator (EQ/BEGINS, and so forth).
- A comma-separated list for each column/value.

Returns: LOGICAL

Notes:

- This procedure is designed to run on the client and to be called possibly multiple times to build up the query's WHERE clause (storing intermediate results in the QueryString property) before it is finally used in a Query–Prepare method.
- openQuery takes care of the preparation of the QueryString property.
- The QueryColumns property is used to ensure that each column and operator will be added only once to the QueryString. The property is also used to store the offset and length of the corresponding values.

assignQuerySelection (sbo.p)

SmartBusinessObject version of this where-clause function. It separates the Columns by SmartDataObject and appropriately passes on the columns, their values, and operators.

Parameters:

`INPUT pcColumns AS CHARACTER`

Comma-separated list of column names.

`INPUT pcValues AS CHARACTER`

CHR(1)-separated list of the corresponding values.

`INPUT pcOperators AS CHARACTER`

Operator (one for all columns):

- A blank defaults to (EQ).
- A slash defines alternative string operators (EQ/BEGINS, and so forth).
- A comma-separated list for each column/value.

Returns: LOGICAL

Notes: All columns must be qualified by their SmartDataObject Objectname as TableName; this will be replaced with RowObject when the columns are passed on to the SmartDataObject.

assignUnsubscribe (consumer.p)

Changes the unsubscribe-on-close flag for a persistent subscription.

Parameters:

```
INPUT pcDestination AS CHARACTER

INPUT pcSubscription AS CHARACTER

INPUT plUnsubscribe AS LOGICAL
```

Returns: Nothing (procedure)

Notes: This gives the application the ability to ask users at logoff whether they would like to cancel a subscription that would otherwise persist into the next session.

bandActions (toolbar.p)

Returns the available menu band for this toolbar master.

Parameters:

```
INPUT pcParentBand AS CHARACTER
```

Returns: CHARACTER

Notes: Used in the Instance Property dialog to select MenuBands

bandSubmenuLabel (toolbar.p)

Returns the label for a specific child band, currently used for the instance-properties dialog.

Parameters:

```
INPUT pcBand AS CHARACTER


INPUT pcChildBand AS CHARACTER
```

Returns: CHARACTER

Notes: Currently used for instance property dialog

bandSubmenus (toolbar.p)

Returns the available child bands on a specific band

Parameters:

`INPUT pcParentBand AS CHARACTER`

Band name. If the empty string, the menubar of the Toolbar master.

`INPUT pcLabels AS LOGICAL`

Return the action labels, if defined

Returns: CHARACTER

Notes: Used in the Instance Property dialog to select MenuBands

batchServices (data.p)

Groups a sequence of SmartDataObject service requests into a single request and thereby minimizes network messaging and improves performance.

Parameters:

`INPUT pcServices AS CHARACTER`

A CHR(1)-delimited list of SmartDataObject internal procedures and functions to be executed. Each entry consists of a CHR(2)-delimited list of INPUT PARAMETER values with the first entry being the NAME of the procedure or function to be executed.

`OUTPUT pcValues AS CHARACTER`

A CHR(1)-delimited list of CHR(2)-delimited strings of output values that result from the execution of the services listed in pcServices (above). There is a one-to-one correspondence between the CHR(1)-delimited list of Services in pcServices (above) and the CHR(1)-delimited list in pcValues. Procedures with no output parameters have a NULL entry. The return values of functions appear as the first entry of the corresponding CHR(2)-delimited list, followed by any output parameters.

Returns: Nothing (procedure)

Notes: As shipped, batchServices supports only a limited list of services. This list consists of all “get” and “set” functions that do not require processing other than getting and setting the property in the ADMProps temp-table record as well as columnProps, initializeObject, openQuery, and synchronizeProperties. The list of cases will be extended from release to release. Developers can extend the list by overriding batchServices with a local version that has an extended list then running SUPER.

beginTransactionValidate (datalogic.i)

Wrapper routine called automatically on entry into a transaction block.

Parameters: None

Returns: Nothing (procedure)

Notes: None

bindServer (appserver.p)

Binds the object to the server, as does getASHandle. However, bindServer does not expose the handle, making it a better choice for outside callers.

Parameters: None

Returns: None (procedure)

Notes: None

blankField (filter.p)

Fully blanks some field and sets MODIFIED to FALSE.

Parameters:

`INPUT phField AS HANDLE`

The handle of the actual filter field.

`INPUT phRangeField AS HANDLE`

The handle of the optional range field.

```
INPUT phOperator AS HANDLE
```

The handle of the optional operator.

Returns: LOGICAL

Notes: Called by blankFields and resetFields. The special blankFillIn function ensures that fill-in widgets are fully blanked.

blankFields (filter.p)

Restores all filter fields to the empty, unmodified state.

Parameters: None

Returns: Nothing (procedure)

Notes: None

blankFillIn (filter.p)

Makes a fill-in appear blank by manipulating its format string.

Parameters:

```
INPUT phFillIn AS HANDLE
```

The handle of the fill-in to be blanked.

Returns: LOGICAL

Notes: Utility function called by blankFields and blankField.

browseHandler (select.p)

Sets the properties in the newly started Browse widget.

Parameters:

`INPUT phBrowseObject AS HANDLE`

Handle of the Browse.

Returns: Nothing (procedure)

Notes: Called by initializeBrowse before the Browse has been linked or initialized. Override this function to set/override properties for the Browse.

bufferCompareDBToRO (query.p)

Performs a BUFFER-COMPARE of a database buffer and a RowObjUpd buffer.

Parameters:

`INPUT phRowObjUpd AS HANDLE`

The update buffer.

`INPUT phBuffer AS HANDLE`

The database buffer.

`INPUT pcExcludes AS CHARACTER`

Comma-separated list of fields not to be compared.

`INPUT pcAssigns AS CHARACTER`

Comma-separated list of field pairs to be compared individually. The pairs are mappings from the target/source buffers where the field names differ.

Returns: LOGICAL

Notes: The primary purpose of this function is to detect when individual array fields are referred to in the assign-list (pcAssigns) from the database buffer and ensure that they are compared properly.

bufferCopyDBToRO (query.p)

Performs a BUFFER–COPY of a database buffer to a row object buffer. In particular, if an assign–list is used to map individual array elements from the database buffer to the row object buffer, this procedure will ensure that the values are copied properly.

Parameters:

`INPUT phRowObj AS HANDLE`

Handle to the row object buffer that is to be the target of the BUFFER–COPY.

`INPUT phBuffer AS HANDLE`

Handle to the database buffer that is to be the source of the BUFFER–COPY.

`INPUT pcExcludes AS CHARACTER`

Comma-delimited list of fields to be excluded from the BUFFER–COPY.

`INPUT pcAssigns AS CHARACTER`

Comma-delimited list of field pairs to be individually copied. The field pairs are mappings of fields from the target/source buffers where the field names differ.

Returns: Nothing (procedure)

Notes: The primary purpose of this procedure is to detect when individual array fields are referenced in the assign–list (pcAssigns) from the database buffer and to ensure that they are copied properly. It is assumed that the database buffer is always the source of the BUFFER–COPY, so that the second field in the assign–list field pair is always where the individual array reference will be found (for example, pcAssigns = “ROfld1,DBfld[1]”).

bufferHandle (action.p)

Returns the handle of the Actions temp-table, if the argument is "Action". Else returns an undefined handle.

Parameters:

`INPUT pcObject AS CHARACTER`

Object for which the handle is sought.

Returns: HANDLE

Notes: None

bufferWhereClause (query.p)

Returns the complete WHERE clause for a specified buffer, including leading blanks and trailing blanks before the period, but excluding commas, the period, and trailing blanks after the period. Returns the empty string if there is no WHERE clause found.

Parameters:

`INPUT pcBuffer AS CHARACTER`

The buffer of interest. See Notes.

`INPUT pcWhere AS CHARACTER`

The query-prepare string, which might be "?" if the original string is meant to be used.

Returns: CHARACTER

Notes:

- Utility function that uses no properties.
- If Target-Procedure is Super, the buffer's qualification must match the query's. If Target-Procedure is not Super, the buffer will be corrected if it exists in the object's query.

buildAllMenus (toolbar.p)

Builds all submenu branches before the persistent trigger Menu-Drop creates them at the mouse click. Thus, accelerators (keyboard shortcuts) will be enabled as from startup.

Parameters: None

Returns: Nothing (procedure)

Notes: None

buildList (select.p)

Builds the list of fields to display and the corresponding SCREEN-VALUE for the widget.

Parameters: None

Returns: CHARACTER PRIVATE

Notes: Loops through all the rows in the data-source, building the list.

buildMenu (toolbar.p)

Builds one branch of a menu.

Parameters:

```
INPUT pcParent AS CHARACTER
```

The name of the submenu that this menu will use as its parent.

Returns: LOGICAL

Notes: Called from onMenuDrop().

calcWidth (browser.p)

Calculates the exact width of the Browse in the dynamic SmartDataBrowser based on the columns being displayed by it.

Parameters: None

Returns: Nothing (procedure)

Notes: Called by initializeObject.

callOutParams (b2b.p)

Looks through a table and calls zero or more methods based on the value of *pdNode*.

Parameters:

```
INPUT pdNode AS DECIMAL
```

Returns: LOGICAL

Notes: None

cancelNew (browser.p)

Deletes a currently selected NEW row from the Browse's viewport.

Parameters: None

Returns: Nothing (procedure)

Notes: Published from dataSource on cancel of a new row. This works for the case of an SBO where the SDO has no updateSource and the SBO has several. Too much work is involved to have some kind of property to keep track of this. Publish seems to be a more future-proof and dynamic way to do this, as it should also work when supporting multiple/switchable updateSources for SDOs. As the method ensures that this really is new, this should ensure that the query only can have one browse.

cancelNew (sbo.p)

Receives the cancelNew event from a contained SDO and passes it on to the appropriate DATA-TARGET.

Parameters: None

Returns: Nothing (procedure)

Notes: None

cancelObject (containr.p)

Cancels an object.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

- If this is the window container or a virtual container then override and do not call SUPER. If not a window/virtual container, cancel and undo all containertargets and then destroy. Published from containerTargets or called directly
- There is a slight overhead in this construct as destroyObject (called from exitObject - > apply ‘close’) does a publish ‘confirmExit’, which really is unnecessary after this has published ‘confirmCancel’. The reason is that destroyObject might be called directly.
- We currently have to call exitObject as AppBuilder’s WAIT-FOR complains if we destroy the object directly. Even apply ‘close’ to target-procedure does not trigger the WAIT-FOR. It seems as this has to be fired from the actual instance. (exitObject should be local in all container instances) This might very well be a problem for application WAIT-FOR as well.

cancelObject (datavis.p)

Passes an cancel request to its container

Parameters: None.

Returns: Nothing (procedure).

Notes: The convention is that the standard routine always passes an cancel, ok, or exit request to its CONTAINER-SOURCE. The container that is actually able to initiate the cancel should override this and **not** call SUPER.

cancelRecord (browser.p)

SmartDataBrowser version of cancelRecord. Cancels an Update, Add, or Copy operation. For a cancel of an add or copy, deletes the previously added row from the RowObject temp–table in the associated SmartDataObject, and redisplays the previously current record. For a cancel of an update, redisplays the field values of the previous row. Sets the value of the NewRecord property back to "No".

Parameters: None

Returns: Nothing (procedure)

Notes:

- Invokes cancelRecord in `datavis.p` (RUN SUPER).
- Invoked when an update is cancelled, usually by choosing the Cancel button in an Update SmartPanel or the Cancel button or Cancel menu item in a SmartToolbar.
- Customized when additional processing is needed when a record Add, Copy, or Update is cancelled. You might want to do this to restore other related nondatabase fields or other objects to their original values, or to undo some action that was taken when the Add, Copy, or Update was initiated.

cancelRecord (datavis.p)

Cancels an Update, Add, or Copy operation. For a cancel of an add or copy, resets the fields in their previous state. If no record was available before the add, the fields are disabled.

Parameters: None

Returns: Nothing (procedure)

Notes:

- Invokes the cancelRow() function in the update target of the visualization to delete the row created for add or copy.
- When you choose the Cancel button during:
  - **Add** or **Copy** — The new record is erased and the previously current record is redisplayed.
  - **Update** — The Update is cancelled and the record's original values are redisplayed.

- Invoked when an update is cancelled, usually by choosing the Cancel button in an Update SmartPanel or the Cancel button.
- Sets DataModified property FALSE.
- Customized when additional processing is needed when a record Add, Copy, or Update is cancelled. You might want to do this to restore other related nondatabase fields or other objects to their original values, or to undo some action that was taken when the Add, Copy, or Update was initiated.

cancelRecord (viewer.p)

Cancels an Update, Add, or Copy operation.

Parameters: None

Returns: Nothing (procedure)

Notes:

- When you choose the Cancel button during:
  - **Add** or **Copy** — The new record is erased and the previously current record is redisplayed.
  - **Update** — The Update is cancelled and the record's original values are redisplayed.
- cancelRecord is invoked when an update is cancelled, usually by choosing the Cancel button in an Update SmartPanel or the Cancel button or Cancel menu item in a SmartToolbar.
- cancelRecord republishes the "cancelRecord" event to handle the cancel for GroupAssign targets.
- cancelRecord can be customized when additional processing is needed when a record Add, Copy, or Update is canceled. You might want to do this to restore other related nondatabase fields or other objects to their original values, or to undo some action that was taken when the Add, Copy, or Update was initiated.

cancelRow (data.p)

Cancels an Add, Copy, or Save operation.

Parameters: None

Returns: CHARACTER

Notes: Calls to cancelRow in turn callUndoUpdate, which restores the original values of a modified row from the RowObjUpd record then deletes the RowObjUpd record. In the cases of an Add or a Copy, both the new RowObject and the RowObjUpd records are deleted. After all of this work has been done, doUndoUpdate repositions the RowObject temp-table to what was previously the current row. This is all done in doUndoUpdate because direct access to the tables in the `data.i` include file greatly simplifies the code.

cancelRow (sbo.p)

SmartBusinessObject version of this function. It passes the cancel request on to the appropriate contained SmartDataObject, which does the actual work.

Parameters: None

Returns: CHARACTER

Notes: See the entry for data.p.

canFindAction (action.p)

Returns TRUE if an action exists.

Parameters:

```
INPUT pcAction AS CHARACTER
```

Name of the action.

Returns: LOGICAL

Notes: None

canFindCategory (action.p)

Calls findCategory to determine whether some category exists in TARGET-PROCEDURE.

Parameters:

```
INPUT pcCategory AS CHARACTER
```

The category (action) of interest

Returns: LOGICAL

Notes: None.

canFindModRow (data.i)

Returns TRUE if some rowIdent value can be found in the ModeRowIdent temp-table.

Parameters:

```
INPUT pcRowIdent AS CHARACTER
```

The RowIdent of interest.

Returns: LOGICAL

Notes: None

canNavigate (data.p)

Returns TRUE if this object has no children or its children have no commits pending. Children with uncommitted updates prevent navigation by the parent.

Parameters: None

Returns: LOGICAL

Notes: This routine publishes isUpdatePending because that includes rowObjectState in the check. Navigation objects receive updateState from the objects they navigate and must perform this check in the source of any updateComplete message. The updateComplete message can come from a branch of a data-link tree; publishing isUpdatePending checks the whole tree.

canNavigate (datavis.p)

Returns TRUE if the object can allow a navigation.

Parameters: None.

Returns: LOGICAL

Notes:

- An object cannot navigate if there are uncommitted or unsaved changes in the updateTarget or unsaved changes in any of the Datatargets of the updatetarget. - The check is used as rule for enabling of toolbar actions. This includes delete and add / copy as navigation is an implicit result of these actions on the client.
- See details in the data.p canNavigate

canNavigate (sbo.p)

Acts as a pass-through for the same function in some contained SDO, passing the result back to the caller to which the SDO is mapped. An SDO can navigate if it has no children or its children have no commits pending. Children with uncommitted updates prevent navigation by the parent.

Parameters: None

Returns: LOGICAL

Notes: This routine publishes isUpdatePending because that includes rowObjectState in the check. Navigation objects receive updateState from the objects they navigate and must perform this check in the source of any updateComplete message. The updateComplete message can come from a branch of a data-link tree; publishing isUpdatePending checks the whole tree.

categoryActions (toolbar.p)

Returns the available actions for a specific category on this toolbar master.

Parameters:

```
INPUT pcCategory AS CHARACTER
```

The category of interest. "*" means “all”; the empty string means “uncategorized”.

Returns: CHARACTER

Notes: Used in the Instance Property dialog to select MenuBands

categoryLink (action.p)

Returns the value of Link for some category.

Parameters:

```
INPUT pcCategory AS CHARACTER
```

The category of interest

Returns: CHARACTER

Notes: None.

changeCursor (smart.p)

Sets the cursor on all windows and on any dialog box frames that are currently on the screen.

Parameters:

```
INPUT pcCursor AS CHARACTER
```

Name of cursor to use. This should be either "WAIT" or "".

Returns: Nothing (procedure)

Notes: Normally used internally by the ADM. Could be used by application code to set and then clear the WAIT (hourglass) cursor during a lengthy operation.

changePage (containr.p)

Views (and creates if necessary) objects on a newly selected page in a Container when CurrentPage is reset.

Parameters: None

Returns: Nothing (procedure)

Notes:

- The changePage procedure is normally used internally, called from either selectPage or viewPage. It can be customized to perform some application-specific task each time a page is changed, but developers must note that if it is called from selectPage (the normal case for switching pages within a Container), the previous page will already have been hidden.
- The changePage procedure expects the CurrentPage property to have been set to the new page before it is called.
- The changePage procedure does not hide the previously selected page. This will be done before it is called from selectPage. If it is called from viewPage, the previous page is not hidden first. This would be the case if the new page is a SmartWindow to be viewed in addition to the current page in its container.
- The changePage procedure first publishes changeFolderPage to let the associated SmartFolder visualization, if any, know about the page change.
- If the new page is not page zero (which is always initialized at startup), and the objects on the page have not yet been created, changePage runs createObjects to create all the SmartObjects on the new page. If the SmartContainer itself has been initialized, it also publishes “initializeObject” to initialize the new objects, and publishes “viewObject” to view them.

Examples:

```
/* Display the current page number in a fill-in field whenever the page is
changed.*/
PROCEDURE changePage:
RUN SUPER. /* Perform the standard code first. */
  FILL-IN-1:SCREEN-VALUE IN FRAME {&FRAME-NAME} =
    STRING(DYNAMIC-FUNCTION("getCurrentPage":U)).
END PROCEDURE.
```

characterValue (b2b.p)

Traps the event of a text node being parsed.

Parameters:

```
INPUT pcPath AS CHARACTER
```

```
INPUT pcValue AS CHARACTER
```

Returns: Nothing (procedure)

Notes: None

checkRule (panel.p)

Check the rule of an action against the target

Parameters:

```
INPUT pcRule AS CHARACTER
```

The rule to be checked

```
INPUT phTarget AS HANDLE
```

The activeTarget

```
INPUT plDefault AS LOGICAL
```

Used when function/property is not found or if the function returns "?".

Returns: LOGICAL

Notes: None.

clearActionCache (action.p)

Clears cached data by emptying the ttCategory and ttAction temp-tables.

Parameters: None.

Returns: Nothing (procedure).

Notes: Called by instance clearCache as well as directly by the Session Manager

clearCache (action.p)

Clears cached data in ttCategory and ttAction by calling clearActionCache.

Parameters: None.

Returns: Nothing (procedure).

Notes: Toolbar override deletes toolbar class tables

clearCache (toolbar.p)

Clear cached data by Emptying temp-tables

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

clearLogicRows (logic.i)

Cleans up tables, including RowObjUpd.

Parameters: None

Returns: Nothing (procedure)

Notes: None

clearToolbarCache (toolbar.p)

Clear cached data by Empty temp-tables

Parameters: None.

Returns: Nothing (procedure).

Notes: Called from instance clearCache but also directly from session manager

clientSendRows (data.p)

Client-side routine called by sendRows when operating in n-tier mode (client, AppServer). Calls its counterpart routine (serverSendRows) on the AppServer to assemble and retrieve a batch of records.

Parameters:

`INPUT piStartRow AS INTEGER`

The RowNum value of the record to start the batch to return. Typically piStartRow is ? as a flag to use pcRowIdent instead of piStartRow.

`INPUT pcRowIdent AS CHARACTER`

The RowIdent of the first record of the batch to return. Can also be "FIRST" or "LAST" to force the retrieval of the first (or last) batch of RowObject records.

`INPUT plNext AS LOGICAL`

TRUE if serverSendRows is to start on the NEXT record offset from piStartRow/piRowIdent. Ignored if pcRowIdent is "FIRST" or "LAST".

`INPUT piRowsToReturn AS INTEGER`

The desired number of rows to return.

`OUTPUT piRowsReturned AS INTEGER`

The actual number of rows returned. This number will always be <= piRowsToReturn.

Returns: Nothing (procedure)

Notes:

- All of the parameters are simply received from the caller and passed through to serverSendRows on the AppServer. If piStartRow is not 0 or ? then pcRowIdent is ignored. plNext is ignored if pcRowIdent is "FIRST" or "LAST".
- The most common use of piRowsReturned is to indicate that the entire result list has been returned when it is less than piRowToReturn.
- Runs in the ASHandle if in an SBO.

closeQuery (data.p)

Closes the RowObject temp-table query (on both the client side and the server side if the SmartDataObject is split), then RUNs SUPER, which invokes closeQuery in `query.p`, to close the database query.

Parameters: None

Returns: LOGICAL

Notes: None

closeQuery (query.p)

Closes the database query.

Parameters: None

Returns: LOGICAL

Notes: closeQuery should not be executed on the client side of a split SmartDataObject because the database query only exists on the server side. No harm will be done, but nothing will be closed.

collectChanges (datavis.p)

Collects the screen values from the screen to assign them to the appropriate record. Values are collected as a character value of a list form.

Parameters:

`INPUT-OUTPUT pcChanges AS CHARACTER`

A CHR(1)-delimited list made of fields changed with their corresponding value.

`INPUT-OUTPUT pcInfo AS CHARACTER`

A comma-delimited list with three entries per object containing the number of fields changed, the page number of the object, and its handle. This allows the proper page to be viewed and focus applied to the proper field in the event of an error.

Returns: Nothing (procedure)

Notes:

- Invoked from updateRecord and published to any GroupAssign target.
- The collectChanges procedure is not intended to be run from application code. If validation of modified field values is required, this should generally be done using one of the methods in the SmartDataObject (under submitRow), rather than in the associated visualization.
- Only values of fields that are modified are collected. (The MODIFIED attribute is used to determine if the field is modified or not.)

columnColumnLabel (dataextcols.p)

Returns the label of some column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

This can take any of three forms: unqualified column name of the RowObject field; column name of the RowObject field qualified with "RowObject"; qualified database field name, if the field is mapped to the SDO.

Returns: CHARACTER

Notes: None

columnColumnLabel (sbo.p)

Returns the Progress Column Label of the column identified by the argument. SmartBusinessObject version of the same SmartDataObject function.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Column name of the RowObject field being queried, qualified by the ObjectName of the SmartDataObject.

Returns: CHARACTER

Notes: If no Column Label has been specified, then the Label is returned.

columnDataType (dataextcols.p)

Returns the Progress data type of a specified column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

A column name. This can be either a RowObject field name or a qualified database field name of a table in the database query.

Returns: CHARACTER

Notes: columnDataType recognizes that pcColumn is a qualified database field name based on the presence of a period (".). If pcColumn is a qualified database field name and columnDataType is called on the client side of the SmartDataObject, an attempt is made to resolve this on the client. The first attempt is by using the mapped SmartDataObject name in a recursive call to columnDataType. If this fails, and there is a ForeignFields property, the corresponding data source field is queried by this function. As a last resort, columnDataType is called on the Server side.

columnDataType (filter.p)

Returns the data type of some column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The name of the column in the Filter-Target.

Returns: CHARACTER

Notes: The field will always appear as type CHARACTER at design time, because there is no target available.

columnDataType (query.p)

Returns the data type of a database column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Database field name. Can be in the form DB.TBL.FLDNM, TBL.FLDNM, or FLDNM.

Returns: CHARACTER

Notes: In the case of a field name with an extent, columnDataType is capable of processing a pcColumn specified with brackets.

columnDataType (sbo.p)

Takes a column name, either unqualified or qualified by the SmartDataObject ObjectName, and returns its data type by passing the request along to the appropriate Data Object. If the column name is not qualified by the SmartDataObject ObjectName, the one with the first matching column name is chosen.

Parameters:

```
INPUT   pcColumn AS CHARACTER
```

RowObject column name, optionally qualified by SmartDataObject ObjectName.

Returns: CHARACTER (String representing the data type of the field.)

Notes: The SmartDataObject columnDataType option of permitting a name qualified by database table name is not supported.

columnDbColumn (query.p)

Returns the qualified database name (*DataBase.Table.FieldName*) mapped to the RowObject column specified in pcColumn.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The RowObject name to look up.

Returns: CHARACTER

Notes: None

columnDbColumn (sbo.p)

SmartBusinessObject version of this function. It returns the qualified database name (DatabaseName.TableName.FieldName) mapped to the RowObject column identified by pcColumn.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

RowObject column name of interest.

Returns: CHARACTER

Notes: None

columnExtent (dataextcols.p)

Returns the Progress extent of a specified column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column name of the RowObject field being queried.

Returns: INTEGER

Notes: None

columnExtent (sbo.p)

Returns the extent of the specified RowObject column. SmartBusinessObject version of the SmartDataObject function.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Name of the column for which the extent is wanted.

Returns: INTEGER

Notes: The SDO columnDataType option of permitting a name qualified by database table name is not supported.

columnFilterTarget (filter.p)

Returns the Filter-Target of some column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The name of the column of interest.

Returns: HANDLE

Notes: Currently, this presumes that all columns have the same Filter-Target. This might be changed. Always use this function to determine the Filter-Target of a specific field.

columnFormat (dataextcols.p)

Returns the Progress format of a specified column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column name of the RowObject field being queried.

Returns: CHARACTER

Notes: None

columnFormat (filter.p)

Returns the format string of some column in the Filter-Target.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The name of the column of interest.

Returns: CHARACTER

Notes:

- If the format is overridden, the string will be found in the CHR(1)-separated FieldFormats property.
- If there is a valid Filter-Target, it will query that object and pass through the string it returns. Otherwise, it will return the evaluation of {&CHARFORMAT}.

columnFormat (sbo.p)

Returns the format of the specified RowObject column. SmartBusinessObject version of the SmartDataObject function.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Name of the column for which the format is wanted.

Returns: CHARACTER

Notes: The SDO columnDataType option of permitting a name qualified by database table name is not supported.

columnHandle (dataextcols.p)

Returns the handle of some RowObject column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column name, possibly qualified with "RowObject", or the qualified database field name, if the field is mapped to the SDO.

Returns: HANDLE

Notes: None

columnHelp (dataextcols.p)

Returns the Progress Help text of a specified column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column name of the RowObject field being queried.

Returns: CHARACTER

Notes: None

columnHelp (sbo.p)

Returns the Help text for the specified RowObject column. SmartBusinessObject version of the SmartDataObject function.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Name of the column for which Help is wanted.

Returns: CHARACTER

Notes: The SDO columnDataType option of permitting a name qualified by database table name is not supported.

columnHelpId (filter.p)

Returns the HelpId of some column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The name of the column in the Filter-Target.

Returns: INTEGER

Notes: The actual value is stored in the FieldHelpIds property.

columnInitial (dataextcols.p)

Returns the initial value for a specified field as a character string with the field FORMAT applied.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column name of the RowObject field being queried.

Returns: CHARACTER

Notes: For example, if the value of a numeric field is 123456789 and the format is "999-99-9999", columnInitial would return a character string containing "123-45-6789".

columnInitial (sbo.p)

Returns the initial value for a specified field as a character string with the field FORMAT applied. SmartBusinessObject version of the SmartDataObject function.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Name of the column for which the initial value is wanted.

Returns: CHARACTER

Notes:

- For example, if the value of a numeric field is 123456789, and the format is "999-99-9999", columnInitial() would return a character string containing "123-45-6789"—the result of imposing the format on the value.
- The SDO columnDataType option of permitting a name qualified by database table name is not supported.

columnLabel (dataextcols.p)

Returns the Progress Column label of a specified column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column name of the RowObject field being queried.

Returns: CHARACTER

Notes:

- If no column-label has been specified for pcColumn, then the Label is returned instead.
- If no column-label and no label have been specified, then the column name is returned.

columnLabel (filter.p)

Returns the label of some column in the Filter-Target or the empty string if no label is defined.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column of interest.

Returns: CHARACTER

Notes: If the label is overridden it will be found in the CHR(1)-separated FieldLabels property.

columnLabel (sbo.p)

Takes a column name, either unqualified or qualified by the SmartDataObject ObjectName, and returns its Label by passing the request along to the appropriate SmartDataObject. If the column name is not qualified by the SmartDataObject ObjectName, the first matching column name is used to identify the recipient SmartDataObject.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

RowObject column name, optionally qualified by SmartDataObject ObjectName.

Returns: CHARACTER

Notes: None

columnLabelDefault (filter.p)

Design-time function. Returns TRUE if the label for some Filter-Target column is overridden.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column of interest.

Returns: LOGICAL

Notes: Only used at design time because columnLabel returns the overridden value.

columnMandatory (sbo.p)

Returns TRUE or FALSE depending on whether a specified RowObject field is mandatory. SmartBusinessObject version of the SmartDataObject function.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Name of the column of interest.

Returns: LOGICAL

Notes: The SDO columnDataType option of permitting a name qualified by database table name is not supported.

columnModified (dataextcols.p)

Returns TRUE if a specified column has been modified in the RowObject record.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column name of the RowObject field being queried.

Returns: LOGICAL

Notes: The only way to do this is to see if the row has been modified and then compare the changed and unchanged copies; there is no MODIFIED attribute on the BUFFER–FIELD.

columnModified (sbo.p)

Returns TRUE or FALSE depending on whether a specified RowObject field has been modified. SmartBusinessObject version of the SmartDataObject function.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Name of the column of interest.

Returns: LOGICAL

Notes:

- There is no actual MODIFIED attribute on the BUFFER-FIELD, so this function works by checking whether the row has been modified and, if so, whether the old and new copies are different.
- The SDO columnDataType option of permitting a name qualified by database table name is not supported.

columnObjectHandle (sbo.p)

Takes an unqualified SmartDataObject column name and returns the procedure handle of the first SmartDataObject in this SmartBusinessObject that has that column name.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Unqualified name of the column of interest.

Returns: HANDLE

Notes: Column Name requests for SmartBusinessObjects should normally be qualified by their SmartDataObject ObjectName. This function is used to identify a SmartDataObject when only the column name is known.

columnOperatorStyle (filter.p)

Returns the operator style for some field, derived either from FieldOperatorStyles or the OperatorStyle property.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column of interest.

Returns: CHARACTER

Notes: At present, the function behaves in a restricted way:

- If the value from FieldOperatorStyles is "Inline", the function returns the OperatorStyle property.
- The value “Explicit” is returned only if OperatorStyle is “Range”.

columnPrivateData (dataextcols.p)

Returns the Progress Private-Data attribute of a specified column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column name of the RowObject field being queried.

Returns: CHARACTER

Notes: None

columnPrivateData (sbo.p)

Returns the PrivateData attribute of the specified RowObject field. SmartBusinessObject version of the SmartDataObject function.

Parameters:

`INPUT pcColumn AS CHARACTER`

Name of the column of interest.

Returns: CHARACTER

Notes: None

columnProperty (sbo.p)

Identifies the SDO in which some column*propname* retrieval function should run, runs the function in that SDO, and returns the value so obtained. Internal utility function.

Parameters:

`INPUT pcColumn AS CHARACTER`

RowObject column name.

`INPUT pcProperty AS CHARACTER`

Property name to retrieve.

`INPUT phTarget AS HANDLE`

Procedure handle.

Returns: CHARACTER PRIVATE

Notes: If the SDO is not uniquely identified (pcColumn qualified by the SDO ObjectName property), the request is passed to the first SDO with a matching column name.

columnProps (data.p)

Returns a list of all requested properties for all the requested columns of the RowObject table. If the pcColList parameter is ”*”, then returns all the columns in the RowObject Column List.

Parameters:

`INPUT pcColList AS CHARACTER`

A comma-separated list of column names for which properties are to be returned.”*” indicates all of the columns in the DataObject.

`INPUT pcPropList AS CHARACTER`

A comma-separated list of properties to be returned. Valid list items are DataType, Extent, Format, Help, Initial, Label, Mandatory, Modified, PrivateData, QuerySelection, ReadOnly, StringValue,ValExp, ValMsg, and Width.

Returns: CHARACTER

Notes: Returns a character string of <col1Props> CHR(3) <col2Props> ... where each <colnProps> is of the form: <columnName> CHR(4) Prop1Value> CHR(4) Prop2Value ...

columnQuerySelection (dataextcols.p)

Returns a CHR(1)- separated string with ALL operators and values that have been added to the Query for this column using the assignQuerySelection method.

Parameters:

`INPUT pcColumn AS CHARACTER`

Column name, either SmartDataObject, RowObject name, or qualified database field name of a table in the query.

Returns: CHARACTER

Notes: The data returned represents the query string, which can represent the current query, but is really what will be used in the next openQuery.

columnQuerySelection (query.p)

Returns a CHR(1)-separated string with ALL operators and values that have been added to the Query for this column using the assignQuerySelection method.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Field name of a table in the query. It should be in the form DB.TBL.FLDNM or TBL.FLDNM, but if it is not qualified, the first table in the TABLES property is assumed.

Returns: CHARACTER

Notes:

- The data returned is derived from the QueryString property, so there is no guarantee that the query has been opened yet.
- This method is overridden in `data.p`, where it is assumed that an unqualified column is a RowObject column.

columnQuerySelection (sbo.p)

Returns CHR(1)-separated string with all operators and values—but not the fieldnames—that have been added to the Query for this column by assignQuerySelection(). SmartBusinessObject version of the SmartDataObject function.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Name of the column of interest, qualified with the ObjectName of the owning SmartDataObject.

Returns: CHARACTER

Notes:

- For example, if the query contains "custnum > 5 and custnum < 9", then this function will return: ">#5#<#9" (the "#" represents the CHR(1) character). The field name "custnum" is not included in the return.
- This override is needed to take a RowObject column as input and pass the mapped database column to the super procedure in `query.p`. The data returned reflects the QueryString/QueryColumns properties maintained by the assignQuerySelection. These values might not have been used in an openQuery yet.

columnReadOnly (dataextcols.p)

Returns TRUE if a specified column is READ–ONLY in the SmartDataObject, that is, if it is not in the list of updateable Columns.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column name of the RowObject temp-table that is being queried.

Returns: LOGICAL

Notes: A visualization can use this dynamically to determine whether the field should be updateable on the screen. Even still, there might be circumstances where it is desirable to have it updateable on the screen but if it is READ–ONLY in the SmartDataObject, any changes sent back to the SmartDataObject will not be made in the RowObject temp-table and, therefore, in the database.

columnReadOnly (sbo.p)

Returns TRUE or FALSE depending on whether the specified column is READ-ONLY in the owning SmartDataObject. SmartBusinessObject version of the SmartDataObject function.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Name of the column of interest, qualified by the ObjectName of the owning SmartDataObject.

Returns: LOGICAL

Notes:

- A column is defined as READ-ONLY if it is not in the list of UpdatableColumns.
- A visualization can use this function dynamically to determine whether the field should be updateable on the screen. There might be circumstances where it is desirable to have some field updateable on the screen, but if the field is READ-ONLY in the SmartDataObject, updates made to the values on the screen will be discarded without notice by the SmartDataObject.
- The SDO columnDataType option of permitting a name qualified by database table name is not supported.

columnStringValue (action.p)

Returns the BUFFER-VALUE of the identified column, if it exists.

Parameters:

```
INPUT pcObject AS CHARACTER

INPUT pcId AS CHARACTER

INPUT pcColumn AS CHARACTER

INPUT phTarget AS HANDLE
```

Returns: CHARACTER

Notes: PRIVATE function

columnStringValue (dataextcols.p)

Returns the STRING–VALUE of a specified column in the SmartDataObject RowObject buffer. The STRING–VALUE is the value of a field expressed as a character string formatted according to the columns FORMAT expression.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column name of the RowObject field being queried.

Returns: CHARACTER

Notes: None

columnStringValue (sbo.p)

Returns the STRING-VALUE of some column in the SmartDataObject RowObject buffer. The STRING-VALUE is a character string formed by imposing the column's FORMAT expression on the column's value. SmartBusinessObject version of the SmartDataObject function.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Name of the column of interest, qualified by the ObjectName of the owning SmartDataObject.

Returns: CHARACTER

Notes: None

columnStyleDefault (filter.p)

Returns TRUE if the column’s style is overridden.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column of interest in the Filter-Target.

Returns: LOGICAL

Notes: Only used at design time because the columnOperatorStyle function returns the overridden value.

columnTable (dataextcols.p)

Returns the name of the table mapped to some RowObject column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The identifier of interest, which can be a column name from RowObject or a qualified column name from a database.

Returns: CHARACTER

Notes: Returns the empty string if not found, "?" if ambiguous. Called on the server as a last resort when dealing with unmapped fields.

columnTable (query.p)

Returns the associated table name, possibly qualified with the database name, for a column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The name of the field of interest. Can take any of the forms DB.TBL.FLD, TBL.FLD, or FLD. Unless fully qualified, matches the first found.

Returns: CHARACTER

Notes:

- Call to ensure and fix a column reference according to the query's use of database qualification.
- This function can handle a bracketed argument.
- `Data.p` calls this function only for qualified fields; it overrides it for unqualified RowObject columns.

columnTable (sbo.p)

Returns the database table name of a column. SmartBusinessObject version of the SmartDataObject function.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Name of the column of interest, qualified by the ObjectName of the owning SmartDataObject.

Returns: CHARACTER

Notes: None

columnTooltip (filter.p)

Returns the tooltip for some column in the Filter-Target.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column of interest.

Returns: CHARACTER

Notes: The actual value is returned from the FieldTooltips property.

columnValExp (dataextcols.p)

Returns the validation expression for a specified column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Unqualified column name of the RowObject field; column name qualified with "RowObject"; qualified database field name, if the field is mapped to the SDO.

Returns: CHARACTER

Notes: None.

columnValExp (sbo.p)

Returns the validation expression for some column in a RowObject temp-table. SmartBusinessObject version of the SmartDataObject function.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Name of the column of interest, qualified by the ObjectName of the owning SmartDataObject.

Returns: CHARACTER

Notes: None

columnValMsg (dataextcols.p)

Returns the Progress Validation message for a specified column of the SmartDataObjects RowObject temp-table.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column name of the RowObject field being queried.

Returns: CHARACTER

Notes: None

columnValMsg (query.p)

Returns the validation message for a specified database column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The name of the field of interest. Can take any of the forms DB.TBL.FLDNM, TBL.FLDNM, or FLDNM. Unless fully qualified, matches the first found.

Returns: CHARACTER

Notes: This function can handle a bracketed argument.

columnValMsg (sbo.p)

Returns the validation message for some column in a RowObject temp-table. SmartBusinessObject version of the SmartDataObject function.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Name of the column of interest, qualified by the ObjectName of the owning SmartDataObject.

Returns: CHARACTER

Notes: None

columnValue (dataextcols.p)

Returns the raw (unformatted) character value of some column in a SmartDataObject RowObject buffer.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Unqualified column name.

Returns: CHARACTER

Notes: This is different from columnStringValue, which returns the formatted value.

columnValue (sbo.p)

Returns the raw (unformatted) character value of some field in a RowObject temp-table. SmartBusinessObject version of the SmartDataObject function.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Name of the column of interest, qualified by the ObjectName of the owning SmartDataObject.

Returns: CHARACTER

Notes: None

columnWidth (dataextcols.p)

Returns the width (in character units) of a specified column in the SmartDataObjects RowObject temp-table.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column name of the RowObject field being queried.

Returns: DECIMAL

Notes:

- The width is settable inside of the column editor when editing the fields of the SmartDataObject. However, its default is based on the number of characters needed to satisfy the FORMAT expression assigned to the field.
- The width can be used as a reference by the visualization, but the visualization does not necessarily use the same width.

columnWidth (filter.p)

Returns the width (in character units) of a specified column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The name of the column in the filter-target.

Returns: DECIMAL

Notes:

- Checks the internal FieldWidths property to see if the width is overridden. If not, it returns the default widths from the DefaultCharWidth or DefaultWidth properties.
- The field widget is not used because this value needs to be read before or while the object is created.

columnWidth (sbo.p)

Returns the width in character units of the named column. SmartBusinessObject version of the SmartDataObject function.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Name of the column of interest, qualified by the ObjectName of the owning SmartDataObject

Returns: DECIMAL

Notes: None

columnWidthDefault (filter.p)

Returns TRUE if the width is overridden.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The name of the column in the filter-target.

Returns: LOGICAL

Notes: Only used in design-time because the columnWidth function returns the overridden value.

colValues (browser.p)

Formats the values in the current row of the SmartDataBrowser query for the specified column list. SmartDataBrowser version of colValues.

Parameters:

```
INPUT pcViewColList AS CHARACTER
```

A comma-separated list of requested columns.

Returns: CHARACTER (Values of the specified columns.)

Notes:

- Only used when the SmartDataBrowser has its own database query defined.
- Passes back a CHR(1)-delimited list with the RowIdent as the first value in all cases and fields values (in the order they are requested).

colValues (data.p)

Returns a CHR(1)-delimited list of values for the requested columns (in pcViewColList) of the current row of the RowObject. The first value is the RowObject ROWID and RowIdent (the RowIDs of the database records from which the RowObject row is derived) separated with a comma. Subsequent values are formatted according to each column's FORMAT expression.

Parameters:

```
INPUT pcViewColList AS CHARACTER
```

A comma-delimited list of RowObject column names.

Returns: CHARACTER

Notes: The form of the first value is: <RowObject ROWID>,<DB Buffer1 ROWID>,<DB Buffer2 ROWID>,... This is used as a key to uniquely identify the row and its origins in the database.

colValues (query.p)

Formats into character strings (using the field format specification) a row of values from the current row of the database Query for the specified column list.

Parameters:

```
INPUT pcViewColLis AS CHARACTER
```

A comma-separated list of column names whose values are to be returned.

Returns: CHARACTER

Notes: Passes back a CHR(1)-separated list of formatted values preceded by the RowIdent code (a comma-separated list of rowids of the database records from which the row is derived) as the first value in all cases.

colValues (sbo.p)

Locates requested columns in contained Data Objects and assembles a list of their values. SmartBusinessObject version of the similar SmartDataObject function.

Parameters:

```
INPUT pcViewColList AS CHARACTER
```

Comma-separated list of columns, qualified by the ObjectNames of their respective owning SmartDataObjects.

Returns: CHARACTER

Notes: If values are requested from only one SmartDataObject, then the RowIdent entry returned as the head of the return value will be the RowIdent from that SmartDataObject (its tt rowid + db rowids). Otherwise, a list is returned of all the tt rowids **without** their db rowids. This will allow submitRow() to send the correct rowid on to each contained SmartDataObject on update.columnWidth (`sbo.p`).

Commit (data.i)

Client-side part of the Commit function. Copies changed records into an update temp-table and sends it to the serverCommit procedure.

Parameters: None

Returns: LOGICAL

Notes:

- Called by commitTransaction, which is called by submitRow. The submitRow code has already reserved an unchanged copy of each record to be updated, in case the update must be undone.
- Invokes procedures in the SmartDataObject itself to manipulate the RowObject temp-tables.

commitTransaction (data.p)

This event procedure receives the Commit message from a Commit panel or other Commit source. It then invokes the Commit function to actually commit the changes. After the Commit function finishes, commitTransaction handles any error messages by calling showDataMessages.

Parameters: None

Parameters: Nothing (procedure)

Notes: A transaction block is opened in serverCommit (called by the commit function) only if the SmartDataObject is running C/S. That is, no transaction block occurs on the client side of a split SmartDataObject.

commitTransaction (sbo.p)

Client-side event procedure that receives the Commit event, gathers up the updates from contained SmartDataObjects, and passes them on to the server.

Parameters: None

Returns: Nothing (procedure)

Notes: None

compareDBRow (query.p)

Does a buffer–compare of the current RowObjUpd row to the database records it is derived from. If any fields have been changed, the name of the first table with changed fields is returned in the RETURN–VALUE of the procedure.

Parameters: None

Returns: Nothing (procedure)

Notes: The dynamic buffer–compare does not return the names of fields that were changed, so this information is not available to put into a message. The procedure returns the first table name that has fields that do not match.

confirmCancel (containr.p)

Make sure unsaved changes are cancelled and uncommitted data are undone before allowing its container to initiate a “destroy” operation.

Parameters:

```
INPUT-OUTPUT plError AS LOGICAL
```

TRUE if the destroyObject is to be cancelled

Returns: Nothing (procedure).

Notes:

- The name confirm is used as it is in family with the other confirm methods, but this does not ask any questions Invoked at the top by cancelObject.
- Only republished if we are a non-virtual and non-window container

confirmCancel (datavis.p)

Verifies that no data will be lost before allowing its container to initiate a destroy operation.

Parameters:

```
INPUT-OUTPUT plError AS LOGICAL
```

If TRUE on output, there is a problem and you should cancel the destroy operation

Returns: Nothing (procedure).

Notes: The name confirm is used as it is in family with the other confirm methods, but this does not ask any questions

confirmCommit (datavis.p)

Checks to be sure there are no unsaved changes before allowing the data-source to start a commitTransaction operation.

Parameters:

```
INPUT-OUTPUT plCancel AS LOGICAL
```

Returns TRUE if the transaction can be committed.

Returns: Nothing (procedure)

Notes: None

confirmCommit (query.p)

Checks the state of all data-targets to see if it can commit. If there are unsaved changes, the I-O parameter should return cancel = TRUE. The visual objects (`visual.p`) will, however, offer the user the opportunity to save/cancelRecord in order to be able to commit.

Parameters:

```
INPUT-OUTPUT pioCancel AS LOGICAL
```

Returns TRUE if the transaction should be cancelled.

Returns: Nothing (procedure)

Notes: None

confirmContinue (datavis.p)

Checks to make sure there are no unsaved or uncommitted data changes before allowing its data-source to start an "Apply filter" operation.

Parameters: None

Returns: LOGICAL

Notes: None

confirmContinue (query.p)

Asks all data-targets whether they have changes pending.

Parameters:

```
INPUT-OUTPUT pioCancel AS LOGICAL
```

Returns TRUE if any data-target reports a pending change.

Returns: LOGICAL

Notes:

- If any data-target reports a pending change, this routine returns pioCancel = TRUE to its caller. The caller then must handle the discrepancy, typically by reporting the unsaved changes to the user and requesting disposition instructions (Save, Discard, etc.).
- This method should be called from any method that might change the result set somewhere in the data-source chain like openQuery or navigation actions. Currently called from the filter-source to see if new criteria can be applied. (Currently the Navigation actions are disabled whenever a state that might disallow continuation is set to TRUE, but if a less modal dialog could be achieved by calling this from one of the fetch methods.)

confirmContinue (sbo.p)

If called with a value of FALSE, asks all contained SDOs whether they have changes pending.

Parameters:

```
INPUT-OUTPUT pioCancel AS LOGICAL
```

Returns TRUE if any SDO reports a pending change.

Returns: LOGICAL

Notes: If any SDO reports a pending change, this routine returns pioCancel = TRUE to the caller. The caller then must handle the discrepancy, typically by reporting the unsaved changes to the user and requesting disposition instructions (Save, Discard, etc.).

confirmDelete (datavis.p)

Hook that allows you to force the user to confirm a pending deletion so that it can be aborted if not intended.

Parameters:

```
INPUT-OUTPUT plAnswer AS LOGICAL
```

Returns: Nothing (procedure)

Notes: None

confirmExit (containr.p)

Passes this event on to its descendents, to check whether it is OK to exit (that is, there are no unsaved changes).

Parameters:

```
INPUT-OUTPUT plCancel AS LOGICAL
```

If field values have been modified and not saved, it returns TRUE and the destroyObject is cancelled.

Returns: Nothing (procedure)

Notes: Invoked from destroyObject.

confirmExit (datavis.p)

Data visualization version of confirmExit. Checks the value of the DataModified and RowObjectState properties to make sure there are not unsaved or uncommitted changes before allowing its container to initiate a "destroy" operation.

Parameters:

```
INPUT plCancel AS LOGICAL
```

If field values have been modified and not saved, it returns TRUE and the destroyObject will be cancelled.

Returns: Nothing (procedure)

Notes:

- Invoked from destroyObject and confirmExit itself.
- A SmartContainer publishes this event to all its container targets before initiating destroyObject. If data are modified in one of its container targets, the destroyObject is cancelled.
- The first object where updates are uncommitted will display the message “Unsaved changes. Exit operation cancelled.”.
- Used internally. Could be customized to add additional checks before allowing an Exit, or to modify the message displayed.

confirmOk (containr.p)

Make sure unsaved changes are saved and uncommitted data is committed before allowing its container to initiate a “destroy” operation.

Parameters:

```
INPUT-OUTPUT plError AS LOGICAL
```

TRUE if the destroyObject is to be cancelled

Returns: Nothing (procedure).

Notes:

- This routine is named ‘confirm’ because it is similar to the other confirm methods, though this does not ask any questions.
- Invoked at the top by okObject.
- Only republished if we are a non-virtual and non-window container

confirmOk (datavis.p)

Make sure unsaved changes are saved and uncommitted data is committed before allowing its container to carry out a destroy operation.

Parameters: None.

Returns: Nothing (procedure).

Notes: The name confirm is used as it is in family with the other confirm methods, but this does not ask any questions

confirmUndo (datavis.p)

Checks to make sure there are no unsaved changes before allowing its data-source to start an undoTransaction operation.

Parameters:

```
INPUT-OUTPUT plCancel AS LOGICAL
```

Returns TRUE if the transaction can be undone.

Returns: Nothing (procedure)

Notes: None

confirmUndo (query.p)

If called with a FALSE argument, publishes confirmUndo all data-targets to see whether any are Modified or in AddMode. Returns TRUE if the undo should be cancelled. The visual objects (`visual.p`) will warn the user that unsaved changes will be cancelled.

Parameters:

```
INPUT-OUTPUT pioCancel AS LOGICAL
```

Returns TRUE if the data-targets agree that undo can proceed.

Returns: Nothing (procedure).

Notes: If called with a TRUE argument, the routine does nothing.

constructMenu (toolbar.p)

Runs from initializeMenu when toolbar instance is first instantiated. Returns TRUE if successful.

Parameters: None

Returns: LOGICAL

Notes: This procedure must only be run ONCE and must be run before buildToolbar.

constructMenuBand (toolbar.p)

Constructs a menu band from Repository data. Returns TRUE if successful.

Parameters:

`INPUT pcBand AS CHARACTER`

The band identifier

Returns: LOGICAL

Notes: The band submenu must exist as a handle

constructObject (containr.p)

RUN from adm-create-objects to RUN a SmartObject, and to establish its parent and initial property settings.

Parameters:

`INPUT pcProcName AS CHARACTER`

The procedure name to run.

`INPUT phParent AS HANDLE`

Handle to its parent.

`INPUT pcPropList AS CHARACTER`

Property list to set.

`OUTPUT phObject AS HANDLE`

The new procedure handle.

Returns: Nothing (procedure)

Notes: None

constructObjectMenus (toolbar.p)

Builds some group of menu bands from Repository data.

Parameters:

`INPUT pcBand AS CHARACTER`

The identifier

`INPUT pcPlaceholder AS CHARACTER`

A placeholder action

`INPUT plBlank AS LOGICAL`

Returns: LOGICAL

Notes: None.

constructToolbar (toolbar.p)

Construct the toolbar from Repository data.

Parameters: None.

Returns: LOGICAL

Notes: This will realize bands read from the repository. Non-repository toolbars are created with createToolbar. Both use createToolbarAction to realize the actions/widgets.

containedProperties (sbo.p)

Returns a CHR(3)-delimited list where the first entry is the passed query parameter. The rest of the entries consists of paired objectNames and CHR(4)-delimited properties. For "plDeep" queries the ObjectName will be colon separated to uniquely identify levels in the tree. The second in each pair is a chr(4)-separated list of values where each entry corresponds to the Propertlist for that object. For example: <pcQueryProps>CHR(3)<ObjectName1>CHR(3)<PropAvalue>CHR(4)<PropBvalue>CHR(3)<ObjectName2>CHR(3)<PropAvalue>CHR(4)<PropBvalue>.

Parameters:

`INPUT pcQueryProps AS CHARACTER`

Comma-separated lists of properties to retrieve from SDOs. Optionally use paired semicolon lists to query different Object types.

`INPUT plDeep AS LOGICAL`

If TRUE, continue query in children of children. If FALSE, query only one level down. This parameter is not currently used.

Returns: CHARACTER

Notes: The FORMAT of the returned string is INTERNAL and intended to be transported as is (across servers) to be passed into assignContainedProperties.

copyColumns (data.p)

Called from copyRow to move column values to the new row.

Parameters:

`INPUT pcViewColList AS CHARACTER`

A comma-delimited list of column names.

`INPUT phDataQuery AS HANDLE`

The handle to the RowObject query.

Returns: Nothing (procedure)

Notes: This procedure exists primarily to remove code from the function copyRow(), to reduce the size of the r-code action segment.

copyRecord (browser.p)

Initiates creation of a new record starting with a copy of the previously displayed record. SmartDataBrowser version of copyRecord.

Parameters: None

Returns: Nothing (procedure)

Notes:

- Invokes copyRecord in `datavis.p` (RUN SUPER), which in turn will invoke the copyRow function in the associated SmartDataObject to add a new row to the RowObject temp–table.
- For a SmartDataBrowser, a new blank row will be inserted below the currently selected one. Values that are copied from copyRecord are not displayed here, they are displayed from the ROW–ENTRY trigger (%DLC%/src/adm2/brsentry.i).
- Sets the BrowseInitted property to FALSE. This property is used in the ROW–ENTRY trigger where the actual display of the copied record takes place to make sure the "initialization logic" for copy is executed once.
- Invoked when a Copy operation is initiated, usually by choosing the Copy button in an Update SmartPanel or in a SmartToolbar.
- Customized when copy needs special processing.

copyRecord (datavis.p)

Initiates creation of a new record, starting with a copy of the previously displayed record.

Parameters: None

Returns: Nothing (procedure)

Notes:

- copyRow() is invoked in the update target, which creates the new RowObject temp-table record and copies the current row to it.
- Displays the previously displayed record’s values and allows data entry. The new record is not actually created until it is committed in serverCommit. See the addRecord reference entry for additional notes.
- Sets NewRecord property to "Copy".
- Invoked when a Copy operation is initiated, usually by choosing the Copy button.
- Customized when copy needs special processing.

copyRecord (viewer.p)

Initiates creation of a new record starting with a copy of the previously displayed record.

Parameters: None

Returns: Nothing (procedure)

Notes:

- copyRecord displays the previously displayed record’s values and allows data entry. The new record is not actually created until it is committed in serverCommit. See the addRecord reference entry for additional notes.
- copyRow() is invoked in the update target, which creates the new RowObject temp-table record and copies the current row to it.
- copyRecord is invoked when a Copy operation is initiated, usually by choosing the Copy button from an Update SmartPanel or the Copy button or Copy menu item from a SmartToolbar.

- copyRecord republishes the “copyRecord” event to handle the Copy operation for any GroupAssign targets.
- copyRecord can be customized when a Copy operation needs special processing.

copyRow (data.p)

Creates a new RowObject temp-table record, copies all of the current row values to it. The return value of this function is a CHR(1)-delimited list of the values of the current row as specified in the input parameter pcViewColList. The first value of this return value is the RowIdent of the newly created row.

Parameters:

```
INPUT pcViewColList AS CHARACTER
```

A comma-separated list of columns whose values are to be returned for the newly created row.

Returns: CHARACTER

Notes: None

copyRow (sbo.p)

Passes the column list on to the contained DataObject that manages that data. SmartBusinessObject version of this function.

Parameters:

```
INPUT pcViewColList AS CHARACTER
```

Returns: CHARACTER

Notes: Qualify all columns or no columns.

countButtons (panel.p)

Walks the widget tree of the frame, counts all the buttons, and sets the ButtonCount property.

Parameters: None

Returns: Nothing (procedure)

Notes: This sets the Panel property ButtonCount. For internal use only.

create3DRule (toolbar.p)

Creates a separator in the form of a rectangle between toolbar actions.

Parameters:

```
INPUT phParent AS HANDLE
```

Handle to the parent Frame.

```
INPUT-OUTPUT piX AS INTEGER
```

The position of the separator in X.

Returns: HANDLE

Notes: None

createAttribute (xml.p)

Creates an Attribute in an element and returns its unique id.

Parameters:

```
INPUT pdOwner AS DECIMAL
```

NodeId of OwnerElement.

```
INPUT pcName AS CHARACTER
```

Name of the attribute.

`INPUT pcValue AS CHARACTER`

AttributeValue.

Returns: DECIMAL

Notes: The attribute is not added to the actual element if pcValue is undefined.

createButton (toolbar.p)

Creates a tool button in the toolbar.

Parameters:

`INPUT phParent AS HANDLE`

Parent frame.

`INPUT-OUTPUT piX AS INTEGER`

In X position - out X + height-p.

`INPUT pcName AS CHARACTER`

The name of the button.

`INPUT pcLabel AS CHARACTER`

The label of the button.

`INPUT pcCaption AS CHARACTER`

The caption of the button.

`IONPUT pcBitmap AS CHARACTER`

The bitmap of the button.

`INPUT plSensitive AS LOGICAL`

Whether the button is sensitive.

Returns: HANDLE

Notes: Creates a persistent trigger ON CHOOSE that runs onChoose(pcName).

createConsumers (consumer.p)

Creates message consumers for normal message retrieval, shutdown message retrieval, and error handling.

Parameters: None

Returns: Nothing (procedure)

Notes: A message consumer for normal message delivery is created for each destination temp-table record. A message consumer is created to receive a shutdown message if the ShutDown property has a value. And a message consumer is created to handle errors.

createControls (smart.p)

Default action for SmartObject-specific initialization of ActiveX Controls. Runs adm–create–controls, an AppBuilder-generated procedure.

Parameters: None

Returns: Nothing (procedure)

Notes: A localization of this behavior should be placed in a procedure called createControls in the SmartObject. The Version 8-style name adm–create–controls for the standard behavior is maintained in order to allow a localization in the same procedure file.

createDocument (router.p)

Creates an empty document.

Parameters: None

Returns: LOGICAL

Notes: Override of `xml.p` that does not delete the current document.

createDocument (xml.p)

Creates an empty document.

Parameters: None

Returns: LOGICAL

Notes: None

createElement (xml.p)

Creates an Element in the document and returns its unique id.

Parameters:

`INPUT pcParent AS DECIMAL`

NodeId of Parent.

`INPUT pcName AS CHARACTER`

Name of the node.

`INPUT pcText AS CHARACTER`

Text. ("?" represents no text.)

Returns: DECIMAL

Notes: None

createField (filter.p)

Create the Filter for one field.

Parameters:

`INPUT phFrame AS HANDLE`

Frame handle.

`INPUT pcName AS CHARACTER`

Name of the column.

`INPUT pcDataType AS CHARACTER`

Datatype to use.

`INPUT pcViewAs AS CHARACTER`

View-as.

`INPUT pcFormat AS CHARACTER`

Format.

`INPUT plEnable AS LOGICAL`

Enable for input.

`INPUT pcTooltip AS CHARACTER`

Tooltip text.

`INPUT piHelpid AS INTEGER`

Help ID.

`INPUT pdRow AS DECIMAL`

Row.

`INPUT pdCol AS DECIMAL`

Column.

`INPUT pdHeight AS DECIMAL`

Height. (If this is an editor, a height > 2 will be used to set inner-lines instead of height, and the editor will have word-wrap and a vertical scrollbar.)

`INPUT pdWidth AS DECIMAL`

Width.

Returns: HANDLE

Notes: PRIVATE function.

createLabel (combo.p)

Creates the label for the Combo Box.

Parameters:

```
INPUT pcLabel AS CHARACTER
```

The label string.

Returns: HANDLE

Notes: This function is separated in order to be able to create the label in design-mode. The label is defined in the parent Frame.

createLabel (filter.p)

Creates the label for the filter-field.

Parameters:

```
INPUT phField AS HANDLE
```

The handle of the filter-field.

```
INPUT pcLabel AS CHARACTER
```

The label of the filter-field.

Returns: HANDLE

Notes: PRIVATE function.

createLabel (lookup.p)

Creates the label for the Lookup.

Parameters:

```
INPUT pcLabel AS CHARACTER
```

The label string.

Returns: HANDLE

Notes: This function is separated in order to create the label in design-mode. The label is defined in the parent Frame.

createLabel (select.p)

Creates the label of the selection. The label is added to the parent frame.

Parameters:

```
INPUT pcLabel AS CHARACTER
```

Returns: HANDLE

Notes:

- This function is separated in order to create the label in design-mode.
- Instance properties:
  - Shows label when label is set from the data-source. The dialog knows the Data-Source, but this super procedure cannot find it because the link is not implemented.
  - EndMove. Moves label accordingly.

createMenuAction (toolbar.p)

Creates a menu and returns its handle.

Parameters:

`INPUT phParent AS HANDLE`

The parent of the new menu.

`INPUT pcAction AS CHARACTER`

The actions in the new menu

`INPUT pcParent AS CHARACTER`

The name of the parent

Returns: HANDLE

Notes: None.

createMenuBar (toolbar.p)

Creates an empty menu bar object on the window and returns its handle.

Parameters: None

Returns: HANDLE

Notes: None

createMenuItem (toolbar.p)

Creates a menu item with an ON CHOOSE trigger defined.

Parameters:

`INPUT phParent AS HANDLE`

Parent frame.

`INPUT pcName AS CHARACTER`

Name of the item.

`INPUT pcCaption AS CHARACTER`

Tooltip for the item.

`INPUT pcAccelerator AS CHARACTER`

Action accelerator.

`INPUT plSensitive AS LOGICAL`

Whether the item is sensitive.

Returns: HANDLE

Notes: Creates a persistent trigger ON CHOOSE that runs onChoose(pcName).

createMenuTable (toolbar.p)

Creates a temp-table for a menu.

Parameters:

`INPUT pcParent AS CHARACTER`

The unique action name of already-created parent. If blank, the table belongs to the menu bar.

`INPUT pcName AS CHARACTER`

A unique name.

`INPUT phTarget AS HANDLE`

TARGET-PROCEDURE

`INPUT pcBefore AS CHARACTER`

The unique action name of an already-created sibling.

Returns: LOGICAL PRIVATE

Notes: This is done before it is built in order to be able to insert actions. Because some disable/enable actions might occur before initialization, some menu records might exist with "*" as parent.

createMenuTable2 (toolbar.p)

Create the temp-table for a menu.

Parameters:

`INPUT pcParent AS CHARACTER`

The unique action name of an existing parent, or the empty string if the new menu is the menu bar

`INPUT pcName AS CHARACTER`

The name of the menu

`INPUT phTarget AS HANDLE`

Where to create it

`INPUT pcBefore AS CHARACTER`

The unique action name of an existing sibling

Returns: HANDLE PRIVATE

Notes: PRIVATE This is done before it is built in order to be able to insert actions. Because some disable and enable actions might take place BEFORE initialization, some tMenu record might exist with "*" as parent.

createMenuToggle (toolbar.p)

Creates a menu item that is toggled with an ON CHOOSE trigger.

Parameters:

`INPUT phParent AS HANDLE`

Parent frame.

`INPUT pcName AS CHARACTER`

The name of the button.

`INPUT pcCaption AS CHARACTER`

The tooltip of the button.

`INPUT pcAccelerator AS CHARACTER`

Action accelerator.

`INPUT plSensitive AS LOGICAL`

Whether the item should be sensitive.

Returns: HANDLE

Notes: Creates a persistent ON VALUE-CHANGED trigger that runs onValueChanged(pcName).

createNode (xml.p)

Creates a Node in the document and returns its id.

Parameters:

`INPUT pdParent AS DECIMAL`

NodeId of Parent.

`INPUT pcName AS CHARACTER`

Name of the node.

`INPUT pcType AS CHARACTER`

Node type (defaults to Element).

Returns: DECIMAL

Notes: None

createObjects (containr.p)

Standard code for running the objects in a Container. It runs the AppBuilder-generated procedure named adm–create–objects for compatibility with Version 8.

Parameters: None

Returns: Nothing (procedure)

Notes:

- This procedure does nothing but run adm–create–objects, the AppBuilder-generated procedure that creates and initializes all the SmartObjects in a Container. This allows both the AppBuilder-generated code and any customization of it to exist in the same SmartContainer procedure, because the customization will be called createObjects.
- This can be customized to add application-specific behavior to the creation of SmartObjects in a Container. Note that createObjects will be run once for each page as it is initialized, so code in a custom version should always check the CurrentPage property to make sure that the custom code is executed for the proper page (0 for general code to be executed exactly once).

Examples:

```
PROCEDURE createObjects:

/* Purpose: Add some extra objects to key pages. Note that because these
   objects are being created by custom code, they will not appear in the
   AppBuilder at design time. */

   DEFINE VARIABLE hWin AS HANDLE NO-UNDO.
   RUN SUPER.  /* Get the other objects on this page created. */
   /* Now create a child window which appears when page 100 is selected. */
   IF DYNAMIC-FUNCTION('getCurrentPage':U) = 100 THEN
   DO:
   RUN constructObject (INPUT 'wChild.w':U,    /* Master file to run */
                        INPUT {&WINDOW-NAME},  /* parent handle      */
                        INPUT '':U,  /* No special Instance Properties */
                        OUTPUT hWin).          /* New object handle */
    /* Add a custom link; note that the Container link is always created
       automatically. */
   RUN addLink (THIS-PROCEDURE, 'Custom':U, hWin).
  END.
  RETURN.
END PROCEDURE.
```

createOperator (filter.p)

Creates the widget to use as operator or fill-in for the range option.

Parameters:

`INPUT phField AS HANDLE`

The handle of the filter field.

`INPUT pcType AS CHARACTER`

The widget type ("radio-set" or "combo-box"; otherwise, a fill-in is created).

`INPUT pcValues AS CHARACTER`

List of operator values for the combo-box or radio-set.

`INPUT pdCol AS DECIMAL`

Column position.

`INPUT pdWidth AS DECIMAL`

Column width.

Returns: HANDLE

Notes: None

createRow (data.p)

Accepts a list of values to create a new row in a RowObject temp-table.

Parameters:

```
INPUT pcValueList AS CHARACTER
```

CHR(1)-delimited list of alternating column names and values to be assigned.

Returns: LOGICAL

Notes:

- If an error occurs, it returns FALSE.
- Committing the changes back to the database is a separate step, which will be invoked from here if AutoCommit is set on.

createRule (toolbar.p)

Creates a rule (separator) menu item.

Parameters:

```
INPUT phParent AS HANDLE
```

Handle of the parent frame.

Returns: HANDLE

Notes: None

createSchemaAttributes (b2b.p)

Returns a new query handle for the schema temp-table to use for navigation through a schema node’s children.

Parameters:

```
INPUT piParentNode AS INTEGER
```

Returns: HANDLE

Notes: At present, the caller is responsible for deleting the object.

createSchemaChildren (b2b.p)

Returns a new query handle for the schema temp-table to use for navigation through a schema node’s children.

Parameters:

```
INPUT piParentNode AS INTEGER
```

Returns: HANDLE

Notes: At present, the caller is responsible for deleting the object.

createSchemaPath (b2b.p)

Returns a new query handle for the schema temp-table to use for navigation through a schema node’s children.

Parameters:

```
INPUT pcPath AS CHARACTER
```

Returns: HANDLE

Notes: At present, the caller is responsible for deleting the object.

createSubMenu (toolbar.p)

Creates a submenu item with an ON MENU-DROP trigger.

Parameters:

`INPUT phParent AS HANDLE`

The parent.

`INPUT pcName AS CHARACTER`

Name of the new submenu.

`INPUT pcCaption AS CHARACTER`

Tooltip for the new submenu.

`INPUT plSensitive AS LOGICAL`

Whether the new submenu is to be made sensitive.

Returns: HANDLE

Notes: Creates a persistent ON MENU-DROP trigger that runs onMenuDrop(pcName).

createText (xml.p)

Creates a Text Node in the document and returns its unique id.

Parameters:

`INPUT pdParent AS DECIMAL`

NodeId of Parent Element.

`INPUT pcText AS CHARACTER`

Returns: DECIMAL

Notes: None

createToolBar (toolbar.p)

Create a toolbar object.

Parameters:

`INPUT pcActions AS CHARACTER`

Comma-separated list of actions or action groups. The action RULE specifies a delimiter.

Returns: LOGICAL

Notes: None

createToolbarAction (toolbar.p)

Create a toolbar button.

Parameters:

`INPUT pcName AS CHARACTER`

The action name. The button will be created even if no action is available in the class.

`INPUT pcBand AS CHARACTER`

Optional band identifier used when creating from Repository data.

`INPUT-OUTPUT piXY AS INTEGER`

Button location (interpreted as X or Y depending on toolbar orientation)

Returns: HANDLE

Notes: Create the local temp-table record and call createButton or create3dRule to create the actual widget.

createToolbar (toolbar.p)

Create a toolbar from non-Repository data

Parameters:

```
INPUT pcActions AS CHARACTER
```

Comma-separated list of actions or action-groups to include. "RULE" means a separator line.

Returns: LOGICAL

Notes: None.

createToolbarBorder (toolbar.p)

Adds a border to the toolbar.

Parameters: None.

Returns: LOGICAL

Notes: None.

currentMappedObject (sbo.p)

Returns the object name that is currently mapped to the calling routine.

Parameters:

```
INPUT phRequester AS HANDLE
```

Returns: CHARACTER

Notes: The value is derived from the ObjectMapping property in the SmartDataObject.

dataAvailable (browser.i)

SmartDataBrowser version of dataAvailable. Repositions the SmartDataBrowser to a specific row in response to a query reposition in its Data source.

Parameters:

```
INPUT pcRelative AS CHARACTER
```

Indicates the relative query position in which to reposition. For the current version of dataAvailable, valid values are “First”, “Last”, “Prev”, “Next”, and “Repos.” The values “Same” and “Different” used by other forms of this procedure are simply passed to the super procedure and do not cause a reposition of the browse.

Returns: Nothing (procedure)

Notes: This routine resides in `browser.i` rather than `browser.p` because it must sometimes use `query.p`, and sometimes `datavis.p`, as its super procedure, depending on whether the SmartDataBrowser has its own database query.

dataAvailable (data.p)

Sets the QueryPosition property for the SmartDataObject when the current record changes in some way. Possible settings for the QueryPosition property are “FirstRecord” (the new record is the first record of the RowObject dataset), “LastRecord” (the new record is the last record of the RowObject dataset, “NotFirstOrLast”, “NoRecordAvailable” or “OnlyRecord”.

Parameters:

```
INPUT pcRelative AS CHARACTER
```

A flag to indicate the state of the current record. It can be SAME, to indicate that the current record has been updated. In that case, DataAvailable is published to notify other objects that need to redisplay the record with its modified values; for example, a SmartDataViewer showing that record. Other objects, such as a SmartDataBrowser that is showing related records (for example, Orders of a modified Customer), ignore the event because the parent customer record has been modified, not repositioned to a different customer. Therefore, this version of dataAvailable will not change anything because the new record is the same as the old current record.

For other values of pcRelative, dataAvailable will RUN SUPER if there are ForeignFields, meaning that this is a dependent query to be reopened; the super procedure `query.p` will do this. Otherwise the SmartDataObject's QueryPosition property is reset depending on the new relative position. It then republishes "dataAvailable" to let other related objects know that the record has changed.

Returns: Nothing (procedure)

Notes: This version of dataAvailable only sets the QueryPosition property to FirstRecord, LastRecord, or NotFirstOrLast. It relies on other procedures to set the other possible values.

dataAvailable (datavis.p)

Data visualization version of dataAvailable. Requests that data columns be displayed in the Data source that the data visualization object is connected to, and displays them.

Parameters:

```
INPUT pcRelative AS CHARACTER
```

A flag that signals whether this is a different row than was previously sent; this version of dataAvailable does not care; it will display the row in any case.

Returns: Nothing (procedure)

Notes:

- Invoked whenever the Data source has a new row.
- Invokes colValues( ) in its Data source and displayFields in itself to display the data.
- Parallel to row–available for Version 8 SmartViewers.

dataAvailable (filter.p)

Subscribed to the filter-target in order to make sure that the filter reflects ForeignFields and values.

Parameters:

```
INPUT pcRelative AS CHARACTER
```

Not used

Returns: Nothing (procedure)

Notes: The pcRelative parameter is necessary in order not to override the ForeignField criteria in the query.

dataAvailable (query.p)

dataAvailable is an event procedure that generates a dependent query dynamically based on the ForeignFields property. This event occurs when a data-source publishes “dataAvailable” because it has been repositioned to a different row in its query, and a dependent object needs to react.

Parameters:

```
INPUT pcRelative AS CHARACTER
```

A flag describing the newly available record. Valid values are:

- “SAME”— The current record is being resent because it has been updated. This procedure ignores this value.
- “VALUE–CHANGED”—A target SmartDataObject has changed its query position. This procedure needs to set the QueryPosition property then change pcRelative to “DIFFERENT” before passing it along to other target procedures so as to appear as if the change occurred in this procedure.
- “DIFFERENT”—The RowObject record has changed
- “FIRST”, “NEXT”, “PREV”, “LAST”—Treated the same as “DIFFERENT” in this version of dataAvailable.

Returns: Nothing (procedure)

Notes: This is a special version of DataAvailable for SmartDataObjects with a dependency on another SmartDataObject. The code is different from that for a Viewer. If there are no ForeignFields, then it is being run because of a reposition in one of its targets (a SmartDataBrowser, normally), in which case it just passes on the event to other targets.

dataAvailable (sbo.p)

Notifies other objects (Data-Targets) that the current row has been changed. If the DataSource is an external object, uses the Foreign Fields property to get the key-field values. Uses the key-field values to reopen the Master object's query.

SmartBusinessObject version of the SmartDataObject routine.

Parameters:

```
INPUT pcRelative AS CHARACTER
```

Flag indicating the state of the record.

Returns: Nothing (procedure)

Notes: See the description of the SmartDataObject return for more information.

dataAvailable (select.p)

Logic to receive data from the data-source when the view-as is a fill-in.

Parameters:

```
INPUT pcMode AS CHARACTER
```

Not used

Returns: Nothing (procedure)

Notes: Use a Modify property to avoid marking the value as changed when setdataValue calls fetchRowident in the data-source in order to display the record. If Modify is TRUE, check the source-procedure because this identifies whether this is called directly from "value-changed," or indirectly from "default-action".

dataObjectHandle (sbo.p)

Given the ObjectName (logical name) of a contained SmartDataObject, returns its handle.

Parameters:

```
INPUT pcObjectName AS CHARACTER
```

Returns: HANDLE

Notes: None

dataSource (b2b.p)

Returns the handle of the data source identified by the argument.

Parameters:

```
INPUT pcName AS CHARACTER
```

Returns: HANDLE

Notes: None

dataValue (filter.p)

Returns the RowObject fieldname of a database fieldname.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

```
INPUT pcValue AS CHARACTER
```

Returns: CHARACTER

Notes: None

dbColumnDataName (query.p)

Returns the RowObject field name of a database field name.

Parameters:

```
INPUT pcDbColumn AS CHARACTER
```

A qualified database field name. It must be in the form *DataBase.Table.FieldName*, or the form *Table.FieldName*.

Returns: CHARACTER

Notes: The passed fieldname must match the SmartDataObject's definition in regards to qualifying with database name or not.

dbColumnHandle (query.p)

Returns the handle of a database column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column whose handle is wanted. Can be unqualified (column), partly qualified (table.column), or fully qualified (database.table.column). If not fully qualified, the first match is returned.

Returns: HANDLE

Notes: Is capable of processing a pcColumn specified with brackets. Used by columnDataType, ColumnValMsg, and ColumnTable.

defaultAction (browser.p)

Runs persistently from default action.

Parameters: None

Returns: Nothing (procedure)

Notes: The trigger is defined in setActionEvent.

defineAction (action.p)

Creates a temp-table entry defining an action, possibly overwriting the existing definition.

Parameters:

```
INPUT pcId AS CHARACTER

INPUT pcColumns AS CHARACTER

INPUT pcValues AS CHARACTER
```

Returns: LOGICAL

Notes: If “Instance” is one of the listed fields, the action is created as an instance class and the target-procedure stored in the temp-table.

defineDestination (consumer.p)

Creates a tDestination temp-table record for use by createConsumers( ) for creating message consumers.

Parameters:

```
INPUT pcDestination AS CHARACTER

INPUT pcColumns AS CHARACTER

INPUT pcValues AS CHARACTER
```

Returns: LOGICAL

Notes: This can be called from an override of processDestinations( ) to explicitly create tDestination temp-table records rather than using instance properties.

defineMapping (b2b.p)

Creates a tMapping temp-table record.

Parameters:

```
INPUT pcName AS CHARACTER

INPUT pcColumns AS CHARACTER

INPUT pcValues AS CHARACTER
```

Returns: LOGICAL

Notes: This can be called from an override of processMappings( ) to explicitly create tMapping temp-table records rather than using instance properties.

deleteComplete (browser.p)

Deletes the currently selected row from the viewport of the browse when that row has been deleted from the RowObject dataset.

Parameters: None

Returns: Nothing (procedure)

Notes: Invoked from the SmartDataObject. The SmartDataBrowser is connected to synchronize the viewport with the current contents of the RowObject temp-table being browsed. If the SmartDataBrowser has its own query and is updatable, it will be invoked from deleteRecord, which in this case does nothing as the row has already been deleted.

deleteComplete (sbo.p)

Receives the deleteComplete event from a contained SDO and passes it on to the appropriate DATA-TARGET.

Parameters: None

Returns: Nothing

Notes: None

deleteDocument (xml.p)

Deletes the document and cleans up all the temp-tables.

Parameters: None

Returns: LOGICAL

Notes: None

deleteMenu (toolbar.p)

Deletes the dynamic menu, including all menu items and submenus.

Parameters: None

Returns: LOGICAL

Notes: None

deleteNode (treeview.p)

Removes an individual node from the tree

Parameters:

```
INPUT pckey AS CHARACTER
```

The unique key that identifies the node

Returns: Nothing (procedure).

Notes: None.

deleteObjects (filter.p)

Deletes all dynamic widgets.

Parameters: None

Returns: LOGICAL

Notes: Called at design-time every time the Instance Properties are changed.

deletePage (containr.p)

Deletes all the objects on the specified page.

Parameters:

```
INPUT piPageNum AS INTEGER
```

Number of page to be purged.

Returns: None (procedure)

Notes:

- The deletePage procedure is not run from any standard ADM code; it can be run from application code to delete a page when no longer needed, or for other reasons.
- The deletePage procedure publishes deleteFolderPage to remove the folder tab for the page, if any.

Examples:

```
ON CHOOSE OF Btn_Delete_Page
DO:
  DEFINE VARIABLE lAnswer AS LOGICAL NO-UNDO.
  DEFINE VARIABLE iPage AS INTEGER NO-UNDO.
  /* Get the current page number, and delete it. */
  iPage = getCurrentPage().
  MESSAGE "OK to delete page" STRING(iPage) + "?"
     VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO UPDATE lAnswer.
  IF lAnswer THEN RUN deletePage(iPage).
END.
```

deleteRecord (browser.p)

Deletes the current record. SmartDataBrowser version of deleteRecord. Invoked when the delete button is chosen in an Update SmartPanel or in a SmartToolbar.

Parameters: None

Returns: Nothing (procedure)

Notes: Invokes deleteRecord in `datavis.p` (RUN SUPER), which in turn will run deleteRow in the associated SmartDataObject to delete the row from the RowObject dataset. Makes sure a row is selected in the SmartDataBrowser before deleting the row. Runs deleteComplete when the SmartDataBrowser has its own database query. Typically customized when you want to add a message to confirm the deletion.

deleteRecord (datavis.p)

Initiates the deletion of the current record in the associated SmartDataObject. Invoked when the delete button is chosen in an Update SmartPanel or in a SmartToolbar.

Parameters: None

Returns: Nothing (procedure)

Notes: Invokes deleteRow( ) in the update target the data visualization object is connected to. If the deleteRow( ) fails, error messages are displayed by invoking showDataMessages( ). Typically customized when you want to add a message to confirm the deletion.

Examples:

```
PROCEDURE deleteRecord:
/* Purpose:     Super Override*/
/* Code placed here will execute PRIOR to standard behavior. */
  MESSAGE “Are you sure you want to delete this record?”
  VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO
  UPDATE lAnswer AS LOGICAL.
  IF lAnswer THEN
    RUN SUPER.
/* Code placed here will execute AFTER standard behavior.    */
END PROCEDURE.
```

deleteRecordStatic (query.i)

Deletes a record from the specified table in the query.

Parameters:

```
INPUT piTableIndex AS INTEGER
```

Returns: LOGICAL

Notes: This function is necessary in order to work around a limitation with the BUFFER-DELETE( ) method for buffer handles. The method will fail if there is delete validation defined on the target table. The workaround is to use a static “DELETE {table}.” statement instead.

deleteRow (data.p)

Submits a row for deletion. Returns FALSE if an error occurs.

Parameters:

```
INPUT pcRowIdent AS CHARACTER
```

The RowIdent of the RowObject temp-table to delete.

Returns: LOGICAL

Notes: If auto-commit is on, the row will immediately be sent back to the database for deletion.

deleteRow (sbo.p)

Passes the rowident on to the contained DataObject that manages that data. SmartBusinessObject version of this function.

Parameters:

```
INPUT pcRowIdent AS CHARACTER
```

Returns: LOGICAL

Notes: None

deleteToolbar (toolbar.p)

Delete the dynamic toolbar and its buttons.

Parameters: None

Returns: LOGICAL

Notes: None

deleteTree (treeview.p)

This procedure can be called to remove nodes from the Tree.

Parameters:

`INPUT phTable AS HANDLE`

The handle of tTreeData temp-table (defined in TVController.i)

Returns: Nothing (procedure).

Notes: This differs from the deleteNode procedure in that the tempTable can specify multiple nodes to delete at one time.

describeSchema (data.p)

Returns a temp-table with a schema description in it, assembled by this routine.

Parameters:

`INPUT pcIndexFieldList AS CHARACTER`

The list of fields of interest

`OUTPUT TABLE-HANDLE hTtSchema`

The handle to the ttSchema table

Returns: Nothing (procedure).

Notes: None.

destroyCombo (combo.p)

Cleanup all dynamically created objects.

Parameters: None.

Returns: LOGICAL

Notes: None.

destroyLookup (lookup.p)

Destroys associated lookups, labels, and buttons.

Parameters: None

Returns: LOGICAL

Notes: None.

destroyObject (action.p)

Destroys a SmartToolbar's instance actions as part of the cleanup process.

Parameters: None

Returns: Nothing (procedure)

Notes: None

destroyObject (appserver.p)

Disconnects the AppServer connection if present before invoking the standard destroy code.

Parameters: None

Returns: Nothing (procedure)

Notes: None

destroyObject (b2b.p)

Overrides in order to delete the schema temp-table.

Parameters: None

Returns: Nothing (procedure)

Notes: None

destroyObject (browser.p)

SmartDataBrowser version of destroyObject. Besides invoking the standard behavior in `datavis.p`, destroyObject informs the SmartDataObject (and any attached SmartDataBrowser) that it is no longer being browsed.

Parameters: None

Returns: Nothing (procedure)

Notes: To ensure that a SmartDataObject can only be browsed by one SmartDataObject at a time, when a SmartDataBrowser is destroyed, destroyObject informs the associated SmartDataObject that it is no longer browsed.

destroyObject (combo.p)

Local override of destroyObject. Deletes dynamic Combo Box objects by calling destroyCombo.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

destroyObject (consumer.p)

This is the message consumer's version of destroyObject. It deletes the message consumers and cancels any durable subscriptions that need to be cancelled.

Parameters: None

Returns: Nothing (procedure)

Notes: None

destroyObject (containr.p)

Destroys a container object

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

destroyObject (data.p)

Override to get rid of the datalogicobject

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

destroyObject (design.p)

Destroy all design-time SDO's

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

destroyObject (lookup.p)

local override of destroyObject; delete created widgets

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

destroyObject (messaging.p)

Deletes the JMS session if no other messaging object instances are using it. This routine is used by all messaging objects.

Parameters: None

Returns: Nothing (procedure)

Notes: None

destroyObject (producer.p)

Deletes the Reply and Error message consumers. SmartProducer’s version of destroyObject.

Parameters: None

Returns: Nothing (procedure)

Notes: None

destroyObject (select.p)

Local override of destroyObject; deletes created widgets.

Parameters: None

Returns: Nothing (procedure)

Notes: None

destroyObject (smart.p)

Cleans up and deletes the current object procedure and its descendents, if any.

Parameters: None

Returns: Nothing (procedure)

Notes:

- Checks first to see if any object is not prepared to be destroyed (for example, if DataModified is set). This is done by publishing the named event “confirmExit”, which is implemented for example in `datavis.p` for visual data objects which can return FALSE if their DataModified property is set, indicating that they have unsaved changes to the current record. Because of this possible error return, application code that runs destroyObject should check ERROR–STATUS:ERROR to see whether the operation succeeded or not.
- The destroyObject procedure runs removeAllLinks to delete all SmartLinks associated with this object.
- The standard ADM construct for a SmartWindow is to have the CLOSE trigger run destroyObject. Therefore the statements `APPLY CLOSE to hSmartWin` and `RUN destroyObject IN hSmartWin` will normally have equivalent results. You should use `APPLY CLOSE` so as to catch any other effects of this event. For other SmartObjects, `RUN destroyObject` is the recommended way to destroy the object.
- All SmartContainers will PUBLISH “destroyObject” to delete all the SmartObject procedures they contain, before destroying themselves.
- The destroyObject procedure could be localized to add a check to do cleanup before a destroy completes, or to stop a destroy event from finishing. (See the confirmExit (containr.p) entry for the standard event procedure for doing this.)

destroyObject (toolbar.p)

Destroys dynamic widgets when the object supporting them is destroyed.

Parameters: None

Returns: Nothing (procedure)

Notes: None

destroyObject (xml.p)

Destroys the document. Override this routine to delete the document and temp-table.

Parameters: None

Returns: Nothing (procedure)

Notes: None

destroySelection (select.p)

Cleans up all dynamically created objects.

Parameters: None

Returns: LOGICAL PRIVATE

Notes: None

destroyServerObject (appserver.p)

Destroys the server object.

Parameters: None

Returns: Nothing (procedure)

Notes:

- Called from unbindServer when ServerOperatingMode = "STATELESS" and ASDivision = "CLIENT".
- Objects that need to keep context overrides this with calls to the server that both destroys and returns context.

destroyServerObject (data.p)

Destroys the server object and retrieves its context.

Parameters: None

Returns: Nothing (procedure)

Notes: Called from unbindServer when stateless.

destroyServerObject (sbo.p)

Fetch context from the server-side procedure.

Parameters: None

Returns: Nothing (procedure)

Notes: unbindServer is the public interface to this procedure.

disable_UI (combo.p)

Purges THIS-PROCEDURE, if persistent.

Parameters: None.

Returns: Nothing (procedure).

Notes: Clean up the user-interface. This routine is usually called after execution completes.

disable_UI (lookup.p)

Disables the user interface and releases memory, typically at the end of a session.

Parameters: None.

Returns: Nothing (procedure).

Notes: Here we cleanup the user interface by deleting dynamic widgets we have created and/or hiding Frames.

disableActions (panel.p)

Disable some list of actions

Parameters:

```
INPUT pcActions AS CHARACTER
```

A comma-separated list of actions to be disabled. "*" means "All".

Returns: LOGICAL

Notes: This function is used internally to turn actions on/off depending on the state. Use modifyDisabledActions or setDisabledActions to override enabling.

disable_UI (select.p)

Disables the user interface parameters.

Parameters: None

Returns: Nothing (procedure)

Notes: Clean up the user interface by deleting dynamic widgets that have been created or by hiding frames, as appropriate. This procedure is usually called when ready to clean up after running.

disableButton (lookup.p)

In certain instances, for example when we modify a specific record, we don’t want a user to use a specific Lookup to find another record and potentially modify another record. For this reason we would just disable the specific Lookup’s button and Browser to prevent the use of the Lookup.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

disableButton (select.p)

Disables a SmartSelect’s controls. Used to prevent finding and perhaps unintentionally changing a second record when an update is pending.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

disableField (combo.p)

Sets SENSITIVE and FieldEnabled to FALSE.

Parameters: None.

Returns: Nothing (procedure).

Notes: The SmartDataViewer container will call this procedure if a widget of type PROCEDURE is encountered in disableFields

disableField (lookup.p)

Disable Lookup field in a Viewer

Parameters: None.

Returns: Nothing (procedure).

Notes: The SmartDataViewer will call this procedure if a widget of type PROCEDURE is encountered in disableFields.

disableField (select.p)

Disables SmartSelection component.

Parameters: None

Returns: Nothing (procedure)

Notes: The SmartDataViewer container will call this procedure if a widget of type PROCEDURE is encountered in disableFields.

disableFields (browser.p)

Sets the READ–ONLY attribute of the browse control to TRUE. This is the SmartDataBrowser version of disableFields.

Parameters:

```
INPUT PcFields AS CHARACTER
```

Acceptable values: "All" or "Create".

Returns: Nothing (procedure)

Notes:

- It is not possible to disable only a subset of columns. The "Create" option is intended to support disabling fields that are enabled only for an Add or Copy operation; this is not yet supported.
- Sets the FieldsEnabled property to FALSE.

disableFields (datavis.p)

Uses the {set} pseudo-function to set the value ‘View’ for ObjectMode.

Parameters: None.

Returns: Nothing (procedure).

Notes: Viewer and browser classes have complete override for disabling. The ObjectMode is also set in updateMode, but users might call this directly.

disableFields (filter.p)

Disables fields in the ENABLED-FIELDS list.

Parameters: None

Returns: Nothing (procedure)

Notes: None

disableFields (viewer.p)

Disables the fields in a list represented by the AppBuilder generated preprocessor value {&ENABLED–FIELDS}.

Parameters:

```
INPUT pcFieldType AS CHARACTER
```

Can have values of “All” or “Create“.

Returns: Nothing (procedure)

Notes:

- disableFields is invoked when a SmartDataViewer’s RowObject fields are to be disabled without disabling other fields or variables in the viewer.
- The ENABLED–FIELDS preprocessor value contains a list of RowObject fields that are enabled in a SmartDataViewer. At initialization time, this is turned into the EnabledHandles property, which stores a list of the widget handles of the enabled fields in the SmartDataViewer’s frame, allowing efficient access to them.
- The pcFieldType parameter can have a value of “All” or “Create”. This allows disabling of all the fields or only fields that were used to create (add/copy) a new record. The “Create” option is not yet supported.
- disableFields re-publishes the ‘disableFields’ event to disable fields in any GroupAssign targets.
- When a field in the list is a SmartObject itself (for example, a SmartField object), disableField is run in the SmartObject to disable it.
- For editor fields, READ–ONLY is turned on rather than making the field insensitive since editors need to be sensitive for scrolling.
- disableFields can be customized when you require additional processing when RowObject fields are disabled. You might want to do this to disable or hide other widgets not associated with RowObject fields.

disableObject (treeview.p)

TreeView version of disableObject.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

disableObject (visual.p)

Disables all enabled objects in a frame, including RowObject fields.

Parameters: None

Returns: Nothing (procedure)

Notes:

- disableObject invokes disableFields to disable RowObject fields. In addition, it disables objects in the frame that are not mapped to RowObject fields, which the disableFields procedure will not do. This allows applications to distinguish between disabling data-related fields (because no row is available for display or for some other reason) from disabling the entire object and all fields in its frame.
- disableObject can be customized when additional processing is needed when a SmartObject is disabled. You might do this to selectively re-enable selected widgets in the SmartObject based on the application state or to disable other related widgets.

disablePagesInFolder (containr.p)

Disables the specified folder pages

Parameters:

```
INPUT pcPageInformation AS CHARACTER
```

The pages to be disabled

Returns: LOGICAL

Notes: None.

disconnectObject (appserver.p)

Disables fields in the ENABLED-FIELDS list

Parameters: None

Returns: Nothing (procedure)

Notes: If Asbound is TRUE, an explicit destroyObject is done on the AppServer to give the object an opportunity to clean up. This procedure is invoked from destroyObject, but can also be run directly to disconnect without exiting.

displayActions (action.p)

Utility procedure to put up a dialog showing all the Actions currently defined.

Parameters: None

Returns: Nothing (procedure)

Notes: Can be executed by selecting displayActions from the ProTools procedure object viewer for the desired SmartContainer.

displayCombo (combo.p)

This is published from the SmartViewer containing the SmartDataField to populate the Combo with the evaluated query data, if the query was successful. It is published from displayfields in the viewer. The queries are initially built in getComboQuery.

Parameters:

```
INPUT TABLE FOR ttDCombo
```

Returns: Nothing (procedure).

Notes: This is designed to allow all combo queries to be built at the cost of only a single appserver call.

displayFields (browser.p)

Displays the current row values by moving them to the screen values of the browse. Invokes displayObjects to display the values of any other objects that are in the SmartDataBrowser.

This is the SmartDataBrowser version of displayFields.

Parameters:

```
INPUT pcColValues AS CHARACTER
```

CHR(1)-delimited list of field values. The first entry in the list represents the RowIdent of the row being displayed.

Returns: Nothing (procedure)

Notes:

- Ordinarily the columns in each row of a browse are displayed automatically just by the opening of the query with which the browse is associated. Therefore this procedure is only invoked after changes that have been made to a row are saved or undone, necessitating the redisplay of the current row.
- For special display (modifying the value of a column in each row of the browse, for example), use the ROW–DISPLAY trigger of the SmartDataBrowser.
- Runs displayObjects to display any object values that are part of the SmartDataBrowser. These would be additional fields which have been placed into the Frame along with the browse widget.

displayFields (viewer.p)

Displays the current row values by moving them to the viewer's frame screen-values.

Parameters:

```
INPUT pcColValues AS CHARACTER
```

CHR(1)-delimited list of the STRING–VALUEs of the requested columns; the first value of this list is the RowIdent value of the corresponding record to be displayed.

Returns: Nothing (procedure)

Notes:

- The pcColValues input parameter contains the field values to be displayed in the SmartDataViewer.
- displayFields is invoked from resetRecord, cancelRecord, copyRecord, addRecord, and from dataAvailable when the data source has a new row.
- displayFields runs displayObjects to display values for nondatabase-related fields.
- When a field is a SmartObject itself (for example, a SmartField object), setDataValue is invoked in the SmartObject to set its value.
- Be aware that no RowObject record is available in a SmartDataViewer, so a customization of displayFields cannot refer to the RowObject buffer. The colValues function can be used to access a list of column values from the Viewer's SmartDataObject if needed, or the columnStringValue function will return the formatted value of a particular field.

Examples:

```
PROCEDURE displayFields:

/*------------------------------------------------------------------------
  Purpose:     Super Override
               This SmartDataViewer gets the handle of its container. If
               this is a window, then the title of the window is changed
               to reflect the current record.
  Parameters:
  Notes:
------------------------------------------------------------------------*/

DEFINE INPUT PARAMETER pcColValues AS CHARACTER NO-UNDO.
  DEFINE VARIABLE hContainerSource  AS HANDLE NO-UNDO.
   /* Code placed here will execute PRIOR to standard behavior. */
  hContainerSource = DYNAMIC-FUNCTION('linkProperty':U,
                             INPUT 'Container-Source':U,
                             INPUT 'ContainerHandle':U).
  RUN SUPER( INPUT pcColValues).
  /* Code placed here will execute AFTER standard behavior.*/
  ASSIGN
    hContainerSource:TITLE =
"Customer : " + RowObject.NAME:SCREEN-VALUE IN FRAME {&FRAME-NAME}
               WHEN VALID-HANDLE(hContainerSource) AND ContainerSource:TYPE =
'WINDOW':U
    .
END PROCEDURE.

PROCEDURE displayFields:

/*------------------------------------------------------------------------
Purpose:     Super Override
             If the value of the country field is 'France' in this
             SmartDataViewer, then make the background color of this
             field red.  Otherwise leave it the its default color.
  Parameters:
  Notes:
------------------------------------------------------------------------*/

DEFINE INPUT PARAMETER pcColValues AS CHARACTER NO-UNDO.
  /* Code placed here will execute PRIOR to standard behavior.  */
  RUN SUPER( INPUT pcColValues).
  /* Code placed here will execute AFTER standard behavior.     */
  RowObject.Country:BGCOLOR IN FRAME {&FRAME-NAME} =
  IF RowObject.Country:SCREEN-VALUE = 'France' THEN 12 ELSE ?.
END PROCEDURE.
```

displayLinks (smart.p)

Utility procedure to put up a dialog showing all the ADM links for a given container object.

Parameters: None

Returns: Nothing (procedure)

Notes:

- Can be executed by selecting displayLinks from the PRO*Tools procedure object viewer for the desired SmartContainer.
- As noted, this is a utility procedure that is not executed by any standard ADM code and is not intended to be called from a SmartObject application.

displayLookup (lookup.p)

Finds and displays a Lookup. This routine is published by the SmartViewer to populate the Lookup with the evaluated query data, if the query was successful. The queries were initially built in getLookupQuery.

Parameters:

```
INPUT TABLE FOR ttLookup
```

Returns: Nothing (procedure).

Notes:

- This allows all lookup queries to be built at the cost of a single AppServer call.
- Note that this is not run at all in ADD mode, unless the Lookup is manually fired.

displayObjects (datavis.i)

Displays values for nondatabase-related fields.

Parameters: None

Returns: Nothing (procedure)

Notes:

- Invoked from displayFields.
- Displays the values of the fields that are listed in the AppBuilder-maintained preprocessor variable {&DISPLAYED–OBJECTS}. These are fill-ins and other frame fields that are not associated with fields in the associated SmartDataObject.
- This procedure is located in the `datavis.i` include file rather than the super procedure `datavis.p` in order to provide access to the DISPLAYED–OBJECTS preprocessor value. Therefore, to override this procedure, it is necessary to define a local version of displayedObjects in an individual visual SmartObject and then define the EXCLUDE–displayedObjects preprocessor, which will remove the standard version from the compilation.

displayRecord (datavis.p)

Displays the current datasource's rows

Parameters: None

Returns: Nothing (procedure)

Notes: None

doBuildUpd (data.i)

Transfers changed rows into the Update Temp-Table and returns it to the caller (the Commit function).

Parameters: None

Returns: Nothing (procedure)

Notes:

- This code must be inside the SmartDataObject itself to allow the BUFFER-COPY operations. They can be specialized by defining like-named procedures in another support procedure.
- For each existing row to be updated, there is already a “before” copy in the RowObjUpd table, so create an “after” row.
- There is already a row in RowObjUpd for each Added/Copied row, so just update it with the latest values.
- There is already a row in RowObjUpd for each Deleted row, so there is no need to do anything for these rows.

doCreateUpdate (data.i)

FINDs the specified row to be updated and saves a copy into the RowObjUpd table, to support Undo. Run from submitRow when it receives a set of value changes from a UI object.

Parameters:

`INPUT pcRowIdent AS CHARACTER`

Encoded key of the row to be updated

`INPUT pcValueList AS CHARACTER`

Chr(1)-delimited list of FieldName/NewValue pairs.

`OUTPUT plReopen AS LOGICAL`

TRUE if the row is new, the result of a COPY or ADD

`OUTPUT pcMessage AS CHARACTER`

Error message, if any

Returns: Nothing (procedure).

Notes: Run from submitRow. Returns error message or " ". If the row is not available in the RowObject temp-table (this would be because the SmartDataObject was not the DataSource) this routine FINDs the database record(s) using the RowIdent key before applying the changes, unless it's a new row.

doEmptyModTable (data.i)

Empties the ModRowIdent temp-table.

Parameters: None

Returns: Nothing (procedure)

Notes: ModRowIdent keeps track of records added/modified since the last call to openQuery(). It contains the rowIdents of records already on the client process. Its purpose is to prevent sending duplicate records in upcoming batches.

doEmptyTempTable (data.i)

Empties the RowObject temp-table when the query is reopened while a transaction is active.

Parameters: None

Returns: Nothing (procedure)

Notes: This routine is only needed when a transaction is active. In other cases, use the faster EMPTY-TEMP-TABLE method.

doReturnUpd (data.i)

Runs from Commit on the client side. Obtains the latest state of the records in RowObjUpd from the server for refreshing data-display objects.

Parameters:

```
INPUT cUndoIds AS CHARACTER
```

List of any RowObject ROWIDs for which changes were rejected by a commit. Takes the form of RowNum/ADM-ERROR-STRING pairs, CHR(3) separators inside the pairs, comma separators between the pairs.

Returns: Nothing (procedure)

Notes:

- If the error string in cUndoIds is "ADM-FIELDS-CHANGED", then another user has changed at least one field value. In this case, RowObjUpd fields will contain the refreshed db values and we pass those values back to the client.
- If not autocommit, also reposition here. Otherwise, the caller has both the rowident and more information, (submitCommit knows whether a reopen is required; deleteRow just uses fetchNext if required).

doUndoDelete (data.i)

Restores deleted rows.

Parameters: None.

Returns: Nothing (procedure).

Notes: This is separated because:

- A failed commit should restore this, otherwise the user has to undo to correct mistakes.
- The regular undo needs to restore these.

doUndoRow (data.i)

Restores the row using the saved image of the unchanged RowObjUpd record.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

doUndoTrans (data.i)

Performs the buffer delete and copy operations needed to restore the RowObject temp-table when an Undo occurs. Added or copied records are deleted, modified and deleted records are restored. The RowObjUpd table is emptied.

Parameters:

`INPUT cUndoIds`

List of any RowObject ROWIDs for which changes were rejected by a commit. Takes the form of RowNum/ADM-ERROR-STRING pairs, CHR(3) separators inside the pairs, comma separators between the pairs.

Returns: Nothing (procedure)

Notes: Called by the event procedure undoTransaction. Runs on the client side.

doUndoUpdate (data.i)

Supports cancelRow by copying the current RowObjUpd record back to the current RowObject record.

Parameters: None

Returns: Nothing (procedure)

Notes: None

editInstanceProperties (smart.p)

Runs the dialog to get run-time property settings.

Parameters: None

Returns: Nothing (procedure)

Notes:

- Generally run by the AppBuilder in design mode to bring up the dialog procedure that has been defined in the object property include file as the ADM–PROPERTY–DLG preprocessor value. Normally there will be one such standard program per template, whose name is specified in the template. This dialog allows application designers to define values for SmartObject properties that are appropriate to assign when an instance of the object is created.
- You can create special InstanceProperty programs for specific types of SmartObjects (a particular SmartDataViewer with extra run-time Properties, for example). No assumptions are made about the structure or functionality of the dialog program except that it sets each of the modified Instance Properties by executing their *set* property functions. The existing Instance Property dialog procedures in the `src/adm2/support` directory can be used as models for building new ones or extending existing ones.
- Application code could invoke this procedure (by adding a double-click trigger to a widget, for example), to allow the InstanceProperty dialog to be invoked at run time.
- This is the Version 9 (ADM2) equivalent of the edit–attribute–list procedure in Version 8.
- Generally run by the AppBuilder in design mode.

Examples:

```
PROCEDURE editInstanceProperties:

/* Purpose: Display a different Instance Property Dialog if this object
   is in a Template. Use the normal dialog if this is in a standard
   Master file. */

DEFINE VARIABLE cInfo AS CHARACTER NO-UNDO.
/* use the AppBuilder API to determine if this instance is in a
   Template. */
RUN adeuib/_uibinfo.p (?, 'HANDLE ':U + STRING(THIS-PROCEDURE), 'TEMPLATE':U,
OUTPUT
cInfo).
IF cInfo = 'yes':U THEN   /* Use a special attribute dialog for
                             templates. */
RUN special.w (INPUT THIS-PROCEDURE).
ELSE                      /* Dispatch standard ADM event. */
  RUN SUPER.
END PROCEDURE.
```

emptyTree (treeview.p)

Clears all nodes on the TreeView.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

enableActions (panel.p)

Enable a list of actions

Parameters:

```
INPUT pcActions AS CHARACTER
```

A comma-separated list of actions to be enabled. "*" means "All".

Returns: LOGICAL

Notes: This function is used internally to turn actions on/off depending on the state. Use modifyDisabledActions or setDisabledActions to override enabling.

enableButton (lookup.p)

If you disable the Lookup button to prevent the user from using it, call this routine to re-enable the button again.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

enableButton (select.p)

Undoes the action of disableButton.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

enableField (combo.p)

Enables a Combo field. The SmartDataViewer will call this procedure if a widget of type PROCEDURE is encountered in enableFields.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

enableField (lookup.p)

Enables a Lookup Field. The SmartDataViewer will call this routine if a widget of type PROCEDURE is encountered in enableFields.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

enableField (select.p)

Enables SmartSelection component.

Parameters: None

Returns: Nothing (procedure)

Notes: The SmartDataViewer Container will call this procedure if a widget of type PROCEDURE is encountered in enableFields.

enableFields (browser.p)

SmartDataBrowser version of enableFields. Enables columns in a SmartDataBrowser by turning off the READ–ONLY attribute for the browse object.

Parameters: None

Returns: Nothing (procedure)

Notes:

- If no Update target is present, or if no record is available, the columns in the browse are not enabled.
- Customized when you require additional processing when RowObject fields are enabled. You might want to do this to enable or hide other widgets not associated with RowObject fields.

enableFields (datavis.p)

Sets the value of ObjectMode to ‘Modify’ if the value is not ‘Update’.

Parameters: None.

Returns: Nothing (procedure).

Notes: Viewer and browser classes have complete override for enabling. The ObjectMode is also set in updateMode, but users might call this directly.

enableFields (filter.p)

Enables fields in the ENABLED-FIELDS list.

Parameters: None

Returns: Nothing (procedure)

Notes: None

enableFields (viewer.p)

Enables fields in the ENABLED-FIELDS list AppBuilder generated preprocessor.

Parameters: None

Returns: Nothing (procedure)

Notes:

- When the frame field is a SmartObject (for example, a SmartField), enableField is run in the SmartObject.The ENABLED–FIELDS preprocessor value contains a list of RowObject fields that are enabled in a SmartDataViewer. At object initialization, this is turned into the EnabledHandles property, which holds a list of the widget handles of the enabled fields in the SmartDataViewer's frame, for efficient access.
- If no TableIO–Source (or GroupAssign–Source connected to a TableIO–Source for SmartDataViewers) is present or no record is available, the fields are not enabled.
- enableFields republishes the "enableFields" event to handle enabling fields in any GroupAssign targets.
- enableFields can be customized when you require additional processing when RowObject fields are enabled. You might want to do this to enable or hide other widgets not associated with RowObject fields.

enableObject (panel.p)

Restores the panel buttons to their previous state whenever the panel is re-enabled.

Parameters: None

Returns: Nothing (procedure)

Notes: The “correct” state is the state that the buttons were in before the SmartPanel was disabled. This might not be the “initial” state. If a SmartPanel has been disabled because the page it is on was hidden, its previous state must be restored when the page is viewed again.

enableObject (treeview.p)

TreeView version of enableObject.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

enableObject (visual.p)

Enables an object. Includes all components except RowObject fields. They are enabled using EnableFields().

Parameters: None

Returns: Nothing (procedure)

Notes:

- enableObject is invoked from initializeObject to enable a SmartObject (different from when a SmartObject’s RowObject fields are enabled with enableFields).
- enableObject can be customized when a SmartObject is enabled, perhaps for initialization that is needed each time a SmartObject is enabled, or to enable other related SmartObjects.
- Note that enableObject and disableObject are not completely opposite in their effects. disableObject always invokes disableFields because it is not meaningful to have a SmartObject disabled while RowObject fields inside it are enabled. On the other hand, enableObject does not invoke enableFields; this is done separately to allow enabling of a SmartObject’s RowObject fields independently of other basic widgets it contains.

enablePagesInFolder (containr.p)

Enables the specified folder pages

Parameters:

```
INPUT pcPageInformation AS CHARACTER
```

The pages to be enabled

Returns: LOGICAL

Notes: None.

endClientDataRequest (data.p)

Contains logic to retrieve data properties from the Appserver after a data request. Both queries and commit are considered as data requests.

Parameters: None

Returns: Nothing (procedure)

Notes: The purpose of this function is to encapsulate the logic for stateless and state-aware requests in one call.

endClientDataRequest (sbo.p)

Contains logic to retrieve data properties from the Appserver after a data request. Both queries and commits are considered as data requests.

Parameters: None

Returns: Nothing (procedure)

Notes: The purpose of this function is to encapsulate the logic for stateless and state-aware requests in one call.

endDocument (b2b.p)

Traps the document-processed event and passes all the collected data that has been stored on other events during the processing.

Parameters: None

Returns: Nothing (procedure)

Notes: Collect all the data to ensure that all the updates are done in the order in which the mapped objects are linked to nodes. It is too early on startEvent and even on endEvent because child nodes end before parent nodes (do not save orderlines **before** the order).

endElement (b2b.p)

Traps the XML-element-processed event. Stub routine meant to be overridden.

Parameters:

`INPUT pcPath AS CHARACTER`

Logical path (not the numbered XPath).

`INPUT pcNameSpace AS CHARACTER`

Namespace (future support).

`INPUT pcName AS CHARACTER`

Name of the element.

`INPUT pcQualName AS CHARACTER`

Name qualified with the namespace prefix.

Returns: Nothing (procedure)

Notes: This is just an empty stub to override. It really belongs in `xml.p`, or `xml.p` should use no-error.

endMove (combo.p)

Moves the label whenever the field is moved in design mode.

Parameters: None.

Returns: Nothing (procedure).

Notes: Defined as PERSISTENT on END-MOVE of the frame of the widget.

endMove (lookup.p)

Move the label when the field is moved, in design-mode/

Parameters: None.

Returns: Nothing (procedure).

Notes: Defined as PERSISTENT on END-MOVE of the frame of the widget.

endMove (select.p)

In design mode, moves the label when the field is moved.

Parameters: None

Returns: Nothing (procedure)

Notes: Defined as PERSISTENT on END-MOVE of the frame of the widget.

endTransactionValidate (datalogic.i)

The optional validation routine called automatically by serverCommit after completing all updates, but before exiting the transaction block.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

enterCombo (combo.p)

Trigger fired on entry into a Combo field to store the current screen value of the displayed field.

Parameters: None.

Returns: Nothing (procedure).

Notes: The typical case is that the displayed field will not be the same as the keyfield. In that case, we can fire logic when the displayed field is changed in the leave trigger to see if the new value keyed in actually exists as a unique record, and if so, avoid having to use the combo.

enterLookup (lookup.p)

Trigger fired on entry of lookup field

Parameters: None.

Returns: Nothing (procedure).

Notes: This trigger stores the current screen value of the displayed field in the Lookup. When the displayed and key fields are not the same, and the displayed field is changed in the leave trigger, code can be run to check whether the value entered actually exists as a unique record. If it does, the Lookup need not be used.

enterSelect (select.p)

Trigger fired on entry of SmartSelect field. Copies the screen value to the object so that the lookup can be avoided if the new value actually exists as a unique record.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

errorHandler (consumer.p)

This is the SmartConsumer's version of initializeObject. It fetches messages to get and display from the SUPER version of ErrorHandler.

Parameters:

```
INPUT phMessage AS HANDLE

INPUT phMessageConsumer AS HANDLE

OUTPUT phReply AS HANDLE
```

Returns: Nothing (procedure)

Notes: None

errorHandler (messaging.p)

Handles asynchronous errors, if any.

Parameters:

```
INPUT phMessage AS HANDLE

INPUT phMessageConsumer AS HANDLE

OUTPUT phReply AS HANDLE
```

Returns: Nothing (procedure)

Notes: This adds the message to the message queue for later retrieval.

errorMessage (action.p)

Displays an error message.

Parameters:

```
INPUT iTable AS INTEGER
```

Returns: LOGICAL (FALSE)

Notes: The object is generally forgiving, but some errors are captured.

excludeColumns (query.p)

Returns a comma-separated list of columns that should be used as the exclude parameter for BUFFER-COPY and BUFFER-COMPARE from the database.

Parameters:

```
INPUT pcError AS CHARACTER
```

The number of the table in the query.

Returns: CHARACTER

Notes:

- This protects the target from being unintentionally overwritten by unmapped and not used columns of the same name that are supposed to be referencing a previously copied or compared table, or nothing at all. Remove all the columns that represent the table copied.
- Excludes calculated fields in the last entry. Used by refetchDbRow, transferDbRow, and compareDbRow.

exitObject (smart.p)

Passes an exit request to its container.

Parameters: None

Returns: Nothing (procedure)

Notes:

- By convention, the standard routine always passes an exit request to its CONTAINER–SOURCE. The container that actually initiates the exit should define a local version and **not** call the standard one. That local exitObject is built into the SmartWindow template. This allows any SmartObject to initiate a destroy operation on its container. When it runs exitObject (perhaps in response to the user pressing a "Done" button), this event is passed up through the Container link hierarchy until an object is found that is at the appropriate level to initiate a destroy of all its contents. As noted, this is normally the SmartWindow, where exitObject does "APPLY 'CLOSE' TO THIS–PROCEDURE", that in turn runs destroyObject.

- exitObject could be customized when you want behavior other than, or in addition to, invoking exitObject in the ContainerSource. As noted, the default localization in SmartWindows is to APPLY CLOSE to initiate a destroyObject sequence. A local version of this, in an object other than a SmartWindow, could do additional custom cleanup. Note that because exitObject is defined in the SmartWindow template, a localization of it in a SmartWindow must exclude the standard code to prevent a duplicate procedure definition.

Examples:

```
/*  This example is taken from the standard code for the "Done" button in
    the AppBuilder palette. If the object containing the button is a
    SmartWindow, then the code executes the standard convention for
    destroying a SmartWindow, APPLY "CLOSE". If not, the code runs
    exitObject to pass the event up to its Container, otherwise the effect
    would be to destroy only the SmartPanel or other SmartObject
    containing the button, which is probably not what is intended. */

&IF "{&PROCEDURE-TYPE}" EQ "SmartWindow" &THEN
     RUN exitObject.
&ELSE
     APPLY "CLOSE":U TO THIS-PROCEDURE.
&ENDIF
```

fetchBatch (data.p)

Transfers another batch of rows from the database query to the RowObject temp-table query, without changing the current record position.

Parameters:

```
INPUT plForwards AS LOGICAL
```

TRUE if you want to retrieve the block of rows that follow the current row, FALSE if you want to retrieve the block preceding the current row.

Returns: Nothing (procedure)

Notes:

- Run from a Browser to get another batch of rows from the database query appended to the RowObject temp-table query (when the browser scrolls to the end and not all rows have been retrieved).
- fetchBatch does some checking and sets up the proper parameters to sendRows, but sendRows is called to do the actual work.

fetchBatch (sbo.p)

Returns the next batch of rows to a browse object, communicating with its SmartDataObject.

Parameters:

```
INPUT plForwards AS LOGICAL
```

TRUE if moving forward in the database, FALSE if backward.

Returns: Nothing (procedure)

Notes: None

fetchContainedData (sbo.p)

Gets a set of data from the server, given a where-clause for (currently) the top-level SmartDataObject. Client-side procedure.

Parameters:

```
INPUT pcObject AS CHARACTER
```

If specified, then fetch data sets from that Object down only.

Returns: Nothing (procedure)

Notes: The WHERE clause for the MasterDataObject can be specified in advance of this call by running setMasterQueryWhere(). The WHERE clause for individual SmartDataObjects can also be set by running the standard functions for that purpose (addQueryWhere(), assignQuerySelection()) in the SmartDataObject handle (retrieved by the dataHandle() function).

fetchContainedRows (sbo.p)

A client-side procedure that gets a batch of data from one SmartDataObject and all the tables from SmartDataObjects lower in the data-link chain or tree.

Parameters:

```
INPUT pcObject AS CHARACTER

INPUT piStartRow AS INTEGER

INPUT pcRowIdent AS CHARACTER

INPUT plNext AS LOGICAL

INPUT piRowsToReturn AS INTEGER

OUTPUT piRowsReturned AS INTEGER
```

Returns: Nothing (procedure)

Notes: If pcObject AS CHARACTER is specified, then this procedure only fetches data sets from that Object on down. Intended for internal use. Called from clientSendRows in the contained SDO.

fetchDataSet (browser.p)

Changes or resets the state of a Browse, depending on which action occurs.

Parameters:

INPUT pcState AS CHARACTER

The character value of one of the supported states. Possible values include: "BatchStart", "BatchEnd", "NextStart", "NextEnd", "PrevStart", "PrevEnd".

Returns: Nothing (procedure)

Notes: Used internally to turn on/off the REFRESHABLE attribute of the browse. Used to avoid flashing when batches of records are fetched, or to apply the SET–REPOSITIONED–ROW method to the browse.

fetchDBRowForUpdate (query.p)

Retrieves (with EXCLUSIVE–LOCK) the database records associated with the current RowObjUpd record, as identified by a list of RowIds in the RowIdent field of that record. If the RowMod field is “D” for Delete, it deletes the database records.

Parameters: None

Returns: Nothing (procedure)

Notes: If one or more of the database records cannot be locked (for any reason, most likely because it is locked by another user, but perhaps because it has been deleted), the table name of the first unlockable record is returned as the return value.

fetchDOProperties (sbo.p)

Retrieves any properties from the server-side SmartBusinessObject and its SmartDataObjects, and sets them in the contained SmartDataObjects on the client.

Parameters: None

Returns: Nothing (procedure)

Notes: Currently retrieves the OpenQuery property, which is set only on the server but needed on both sides.

fetchFirst (data.p)

Repositions the RowObject temp-table to the first record, or to the row matching the QueryRowIdent property, if set. If the first record has not yet been fetched (from the database), it calls sendRows to get the first batch of RowObject records of the SmartDataObject, then it repositions the RowObject Temp-Table to the first row.

Parameters: None

Returns: Nothing (procedure)

Notes: None

fetchFirst (query.p)

Repositions the database query to the first row.

Parameters: None

Returns: Nothing (procedure)

Notes: Requests rows to be transferred from the database query if the RowObject query is empty.

fetchFirst (sbo.p)

Retrieves the first row in one of the ContainedDataObjects. It uses the ObjectMapping SmartBusinessObject property to match the caller to its Target, or uses the MasterDataObject by default. It is a SmartBusinessObject version of this event procedure.

Parameters: None

Returns: Nothing (procedure)

Notes: Uses the MasterDataObject by default.

fetchLast (data.p)

Repositions the RowObject query to the last row of the dataset. If the last row has not yet been retrieved from the database, then fetchLast gets the last batch of RowObject records for the SmartDataObject, and then repositions the RowObject query to the last row.

Parameters: None

Returns: Nothing (procedure)

Notes:

- If the SmartDataObject property RebuildOnReposition is false, and the last row from the database query has not yet been fetched, fetchLast keeps asking for batches of rows until the last batch is received. This is required, otherwise there would be a discontinuous set of rows in the RowObject temp-table.
- If RebuildOnReposition is true, and the last row from the database query has not yet been fetched, all RowObject records are discarded and just the last batch is fetched. In this case, the RowNum of the last row becomes 2000000 (just to start with a high value so that all RowNum values will continue to be positive integers), and all other records have smaller numbers (except for additions.)

fetchLast (query.p)

Repositions the Query to the last row.

Parameters: None

Returns: Nothing (procedure)

Notes: None

fetchLast (sbo.p)

Retrieves the last row in one of the ContainedDataObjects. It uses the ObjectMapping SmartBusinessObject property to match the caller to its Target or uses the MasterDataObject by default. It is a SmartBusinessObject version of this event procedure.

Parameters: None

Returns: Nothing (procedure)

Notes: Uses the MasterDataObject by default.

fetchMessages (smart.p)

Returns a delimited list of all messages in their raw form. The message log is cleared.

Parameters: None

Returns: CHARACTER (specially formatted message string)

Notes:

- The fetchMessage procedure is not normally expected to be used by application code. The showDataMessages function can be used—and customized if desired—to parse this specially formatted message string into a series of error messages.
- The message list is delimited by CHR(3); within each message, the Message Text, the Field (if any), and the Table (if any) are delimited by CHR(4).
- As noted, the fetchMessages function clears the message log; it is therefore the responsibility of the calling procedure to display the messages or otherwise deal with them. Use the similar function reviewMessages to read messages without deleting them.

fetchNext (data.p)

Repositions the RowObject query to the next row. If a new batch is required to do so, then sendRows is called to get the new batch.

Parameters: None

Returns: Nothing (procedure)

Notes: None

fetchNext (query.p)

Repositions the Query to the next row.

Parameters: None

Returns: Nothing (procedure)

Notes: None

fetchNext (sbo.p)

Retrieves the next row in one of the ContainedDataObjects. It uses the ObjectMapping SmartBusinessObject property to match the caller to its Target, or uses the MasterDataObject by default. SmartBusinessObject version of this event procedure.

Parameters: None

Returns: Nothing (procedure)

Notes: Uses the MasterDataObject by default.

fetchPrev (data.p)

Repositions the RowObject query to the previous row. If a new batch is needed to do so, then sendRows is called to get the new batch. (Getting a new batch will only be necessary when the RebuildOnReposition property is set to true.)

Parameters: None

Returns: Nothing (procedure)

Notes: None

fetchPrev (query.p)

Repositions the Query to the previous row.

Parameters: None

Returns: Nothing (procedure)

Notes: None

fetchPrev (sbo.p)

Retrieves the previous row in one of the ContainedDataObjects. It uses the ObjectMapping SmartBusinessObject property to match the caller to its Target, or uses the MasterDataObject by default. SmartBusinessObject version of this event procedure.

Parameters: None

Returns: Nothing (procedure)

Notes: Uses the MasterDataObject by default.

fetchRow (data.p)

Repositions the Row Object’s query to the desired row (indicated by piRow) and returns that row’s values (as specified in pcViewColList). The return value is a CHR(1)-delimited list of values (formatted according to each columns FORMAT expression). The first value in the list is the RowIdent of the fetched row.

Parameters:

`INPUT piRow AS INTEGER`

The desired row number within the result set.

`INPUT pcViewColList AS CHARACTER`

A comma-separated list of names of columns to be returned.

Returns: CHARACTER

Notes: None

fetchRowIdent (data.p)

Repositions the RowObject's query to the desired row, based on the corresponding database record ROWID (specified by pcRowIdent), and returns that row's values (as specified in pcViewColList). The return value is a CHR(1)-delimited list of values (formatted according to each column's FORMAT expression.) The first value in the list is the RowIdent of the fetched row. If that row is not in the RowObject table, it repositions the database query to that row and resets the RowObject table.

Parameters:

`INPUT pcRowIdent AS CHARACTER`

The desired rowids within the result set, expressed as a comma-separated list of database rowids.

`INPUT pcViewColList AS CHARACTER`

Comma-separated list of names of columns to be returned.

Returns: CHARACTER

Notes: If called with unknown or nonexisting rowid the query is closed and the SDO will be in a bad state (fetchNext and Prev do not work). The application needs to call fetchRowident with a valid value or fetchFirst, fetchLast, or openQuery to get back to normal.

fieldLookup (filter.p)

Finds the LOOKUP position of the corresponding value to the field in a CHR(1) list of fields and values.

Parameters:

`INPUT pcField AS CHARACTER`

Fieldname.

`INPUT pcList AS CHARACTER`

CHR(1)-separated list with fieldnames and values.

Returns: INTEGER PRIVATE

Notes: PRIVATE used for all defined or overridden column properties.

fieldProperty (filter.p)

Finds the actual value of the corresponding value to the field in a CHR(1) list of fields and values.

Parameters:

```
INPUT pcList AS CHARACTER
```

Fieldname.

```
INPUT pcField AS CHARACTER
```

CHR(1)-separated list with fieldnames and values.

Returns: CHARACTER PRIVATE

Notes: PRIVATE used for all defined or overridden column properties.

filterActive (browser.p)

Calls updateState in the toolbar to set the filter tick on or off, depending on the argument.

Parameters:

```
INPUT plActive AS LOGICAL
```

TRUE if the filter tick is to be turned on.

Returns: Nothing (procedure).

Notes: None.

filterContainerHandler (query.p)

Adds the Filter link between itself and a Filter container. It is called from startFilter after the Filter container is constructed.

Parameters:

```
phFilterContainer AS HANDLE
```

Handle of the Filter container.

Returns: Nothing (procedure)

Notes: The code to add the Filter link has been separated from startFilter so that filterContainerHandler can be overridden, and other links between this object and the Filter container can be added.

filterState (toolbar.p)

Published from NAVIGATION-TARGET to tell the panel to enable filter action when it is linked.

Parameters:

```
INPUT pcState AS CHARACTER
```

Valid value: "FilterAvailable".

Returns: Nothing (procedure)

Notes: None

findAction (action.p)

Returns TRUE if some action exists.

Parameters:

```
INPUT pcAction AS CHARACTER
```

Action of interest.

```
INPUT phTarget AS HANDLE
```

Search space.

Returns: LOGICAL PRIVATE

Notes: Checks for a class action first.

findCategory (action.p)

Calls FIND FIRST on the ttCategory table to determine whether some category pcCategory exists.

Parameters:

```
INPUT pcCategory AS CHARACTER
```

The category of interest

```
INPUT phTarget AS HANDLE
```

NOT USED AT PRESENT

Returns: LOGICAL

Notes: None.

findDataRow (b2b.p)

Finds a row of a mapped DataObject. If any of the KeyFields are blank, use ForeignFields from the object's DataSource.

Parameters:

```
INPUT phObject AS HANDLE
```

The DataObject who's row to find in pcKeyValues

```
INPUT pcKeyValues AS CHARACTER
```

CHR(1)-separated list with the exact same number of entries as the KeyFields property of the object.

Returns: LOGICAL

Notes: None

findRow (data.p)

Finds a row and repositions to it using a key.

Parameters:

```
INPUT pcKeyValues AS CHARACTER
```

Comma- or chr(1)-separated list of keyfields.

Returns: LOGICAL

Notes: None

findRowObjUpd (data.p)

Returns TRUE if the RowObjUpd buffer corresponding to the RowObject record is AVAILABLE.

Parameters:

```
INPUT phRowObject AS HANDLE
```

```
INPUT phRowObjUpd AS HANDLE
```

Returns: LOGICAL

Notes: Called from deleteRow to check for a RowObjUpd for a row being deleted before creating a new one.

findRowWhere (data.p)

Finds a row and repositions to it.

Parameters:

`INPUT pcColumns AS CHARACTER`

Column names (comma separated); fieldname of a table in the query in the form of TBL.FLDNM or DB.TBL.FLDNM (only if qualified with db), (RowObject.FLDNM should be used for SDOs). If the fieldname is not qualified, it checks the tables in the TABLES property and assumes the first is a match.

`INPUT pcValues AS CHARACTER`

Corresponding Values (CHR(1) separated).

`INPUT pcOperators AS CHARACTER`

The operator (one for all columns):

- Blank—Defaults to (EQ).
- Slash—used to define an alternative string operator (EQ/BEGINS, etc.).
- A comma-separated list for each column/value.

Returns: LOGICAL

Notes: The logic is in the `query.p` super.

firstBufferName (query.p)

Returns the first buffer reference in a where-clause expression.

Parameters:

`INPUT pcExpression AS CHARACTER`

A string expression.

Returns: CHARACTER PRIVATE

Notes: In order to be recognized as a buffer name, it must contain at least one ".", otherwise the return value is "?".

firstRowIds (data.p)

Returns the ROWID (converted to a character string) of the first database query row satisfying the passed query prepare string.

Parameters:

```
INPUT pcQueryString AS CHARACTER
```

A complete query WHERE clause that matches the database query’s buffers.

Returns: CHARACTER

Notes: Used by rowidwhere, findRow, and findRowWhere.

firstRowIds (query.p)

Returns the ROWID (converted to a character string) of the first database query row satisfying the passed query prepare string.

Parameters:

```
INPUT pcQueryString AS CHARACTER
```

A complete query WHERE clause that matches the database query's buffers.

Returns: CHARACTER

Notes: Used by rowidwhere, findRow, and findRowWhere.

fixQueryString (smart.p)

Conventionalizes decimal delimiters in query strings to be full-stops.

Parameters:

```
INPUT pcQueryString AS CHARACTER
```

The string to be conventionalized

Returns: CHARACTER

Notes: Wherever a query prepare is being used, call this routine immediately to resolve conventional differences in the query string, such as decimal formatting. The main issues arise when the query string contains stringed decimal values.

formattedValue (select.p)

Returns the formatted value of a passed value according to the SmartSelects format.

Parameters:

```
INPUT pcValue AS CHARACTER
```

The value that needs to be formatted.

Returns: CHARACTER

Notes: Used internally in order to ensure that unformatted data can be applied to screen-value (setDataValue), or used as a lookup in list-item-pairs in (getDisplayValue).

genContextList (data.p)

Generates a list of properties and their values needed to save (and restore) the context of a stateless SmartDataObject between invocations.

Parameters:

```
OUTPUT pcContext AS CHARACTER
```

The context string to be returned. It is a CHR(3)-delimited list of propertyCHR(4)Value pairs.

Returns: Nothing (procedure)

Notes: Currently there are only five properties needed to restore the context: CheckCurrentChanged, RowObjectState, FirstResultRow, LastResultRow, and QueryRowIdent.

hasActiveAudit (browser.p)

If there is a valid DataSource, return the result of hasActiveAudit.

Parameters: None.

Returns: LOGICAL

Notes: None.

hasActiveAudit (data.p)

Parameters: None.

Returns: LOGICAL

Notes: None.

hasActiveComments (browser.p)

If there is a valid DataSource, return the result of hasActiveComments.

Parameters: None.

Returns: LOGICAL

Notes: None.

hasActiveComments (data.p)

Parameters: None.

Returns: LOGICAL

Notes: None.

hasActiveGaTarget (panel.p)

Checks if any group assign targets are active.

Parameters:

```
INPUT phObject AS HANDLE
```

phObject – Procedure object that is tableioTarget and potential GroupAssignTarget.

Returns: LOGICAL PRIVATE

Notes: None

hasForeignKeyChanged (data.p)

Returns TRUE if the dataSource Foreign fields are different from the current ForeignValues.

Parameters: None

Returns: LOGICAL

Notes: Used by SBO commitTransaction.

hasOneToOneTarget (data.p)

Returns TRUE if this DataObject has DataTargets that are updated as part of this.

Parameters: None

Returns: LOGICAL

Notes: None

hideObject (combo.p)

Uses the {get} pseudo-function to get the value of LabelHandle, hides the label, and finally calls RUN SUPER.

Parameters: None

Returns: Nothing (procedure)

Notes: For sizing reasons, the label is not really a part of the object, but added as a text widget to the parent frame.

hideObject (lookup.p)

Uses the {get} pseudo-function to get the value of LabelHandle, hides the label, and finally calls RUN SUPER.

Parameters: None

Returns: Nothing (procedure)

Notes: For sizing reasons, the label is not really a part of the object, but added as a text widget to the parent frame.

hideObject (select.p)

Overrides to ensure that the label is hidden.

Parameters: None

Returns: Nothing (procedure)

Notes: For sizing reasons, the label is not really a part of the object, but added as a text widget to the parent frame.

hideObject (smart.p)

Hides the current object if it is a visual object and sets the ObjectHidden property to true to indicate the state of the object.

Parameters: None

Returns: Nothing (procedure)

Notes:

- The “Hide” concept is a logical one here; nonvisual objects might also be “hidden”, meaning that they are not currently active; this might affect whether code in some event procedures is executed. For example, SmartDataObjects will not respond to events such as fetchNext and fetchLast if they are logically “hidden”, even though they have no visualization. This allows certain SmartLinks to be effectively deactivated when the object at one end or the other is hidden.

- When a SmartContainer is hidden, it is not necessary for all of the objects it contains to be individually hidden because they will be hidden along with the container. For this reason, hideObject, when executed for a SmartContainer, will set the property ContainerHidden in each contained SmartObject without actually running hideObject in each object. The setContainerHidden function will, in turn, set the ObjectHidden property to TRUE so that it can be queried successfully. Not running hideObject in each individual SmartObject improves performance when SmartObjects are being paged (alternately hidden and viewed), and can eliminate problems with “flashing” of visual objects or problems restoring proper frame order when objects are hidden and viewed.

- The hideObject procedure can be localized when behavior in addition to the default is needed when a SmartObject is hidden, for example to decide whether a hidden component should actually be destroyed to conserve memory.

Examples:

```
ON CHOOSE OF MENU-ITEM mi_Hide_Children
DO:
  DEFINE VARIABLE cHandles AS CHARACTER NO-UNDO.
  DEFINE VARIABLE hSMO     AS HANDLE    NO-UNDO.
  DEFINE VARIABLE i        AS INTEGER   NO-UNDO.
  /* Find all the children of a SmartWindow and HIDE the child windows. */
  CHandles = dynamic-function('linkHandles':U,
    INPUT 'Container-Target':U).
  DO i = 1 TO NUM-ENTRIES (cHandles):
    /* cHandles is a comma-separated list of SmartObject handles. */
    hSMO = WIDGET-HANDLE(ENTRY(i, cHandles)).
    /* See if this SmartObject is a SmartWindow. */
     IF dynamic-function('getObjectType':U IN hSMO) = "SmartWindow":U THEN
      RUN hideObject IN hSMO.
  END.
END.
```

hideObject (toolbar.p)

Hides the object.

Parameters: None

Returns: Nothing (procedure)

Notes: For sizing reasons, the label is not really a part of the object, but added as a text widget to the parent frame.

hidePage (containr.p)

Cycles through all objects on a page, hiding them.

Parameters: None

Returns: Nothing (procedure)

Notes: None

imageName (toolbar.p)

Returns the name of some button image.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action of interest

```
INPUT piNumber AS INTEGER
```

The image number, which can be 1 or 2.

Returns: CHARACTER

Notes: This is published thru the container link and are used mainly to validate that a false value received in updateActive can be used to set UpdateActive. This is NOT intended to be called directly, but part of the logic that updates UpdateActive. These are the steps: (1) Updating objects publishes updateActive (true or false) to their container targets. (2) If the value is FALSE the container turns around and publishes this to ALL ContainerTargets before it is stored in UpdateActive. This way the value is only stored as false if ALL contained objects are inactive.

indexInformation (query.p)

Returns indexInformation for all buffers in the query. Each index is separated with CHR(1). Field information is either qualified with db and table or the CHR(2) is used as a table separator.

Parameters:

```
INPUT pcQuery AS CHARACTER
```

```
INPUT plUseTableSep AS LOGICAL
```

```
INPUT pcIndexInfo AS CHARACTER
```

Returns: CHARACTER

Notes: None

initAction (action.p)

Defines all default actions for the SmartToolbar.

Parameters: None

Returns: Nothing (procedure)

Notes: Called once from initializeObject

initDataObjectOrdering (sbo.i)

Initializes the mapping of the AppBuilder-generated order of Upd tables to the developer-defined update order.

Parameters: None

Returns: CHARACTER

Notes: This is used in commitTransaction and serverCommitTransaction to order the table parameters properly.

initializeBrowse (lookup.p)

Initializes the lookup browser window.

Parameters: None

Returns: Nothing (procedure)

Notes: None

initializeBrowse (select.p)

Code to initialize a browse. Starts a browse to show the Data-source data. The browse procedure is specified in the BrowseProcedure property and defaults to the dynamic SmartDataBrowser. A dynamic SmartWindow (BrowseContainer) is started first and the browse procedure is started with the containers constructObject.

Parameters: None

Returns: Nothing (procedure)

Notes: None

initializeCombo (combo.p)

Runs as part of initializeObject to set up the combo.

Parameters: None

Returns: Nothing (procedure)

Notes: Defaults keyfield to actual field if not defined. Defaults displayed field to keyfield if not defined. Assumes a datatype of decimal for keyfield and character for displayed field if different.

initializeLogicObject (datalogic.i)

Loads the data logic procedure as a super procedure.

Parameters: None

Returns: Nothing (procedure)

Notes: None

initializeLookup (lookup.p)

Runs as part of initializeObject to set up the lookup

Parameters: None

Returns: Nothing (procedure)

Notes: Defaults keyfield to actual field if not defined. Defaults displayed field to keyfield if not defined. Assumes a datatype of decimal for keyfield and character for displayed field if different.

initializeObject (action.p)

Calls initAction to define the set of actions used by SmartToolbars.

Parameters: None

Returns: Nothing (procedure)

Notes: None

initializeObject (b2b.p)

Loads schema.

Parameters: None

Returns: Nothing (procedure)

Notes: None

initializeObject (browser.p)

SmartDataBrowser version of initializeObject. Part of the general initialization process. Sets certain property values.

Parameters: None

Returns: Nothing (procedure)

Notes:

- Invokes initializeObject in `datavis.p` (RUN SUPER).
- Starting with Version 9.1A, dynamically creates columns of the SmartDataBrowser when the Dynamic SmartDataBrowser is used.
- For a SmartDataBrowser that is connected to a SmartDataObject as its Data source, sets the DataQueryBrowsed property to TRUE indicating the SmartDataObject is being browsed, and preventing it from being browsed by more than one SmartDataBrowser at a time.
- Sets the value of EnabledHandles, CreateHandles and FieldHandles property.
- Might create a "Search Field" for searching in the SmartDataBrowser if the corresponding SearchField property is set (also new as part of dynamic SmartDataBrowser support for Version 9.1).

initializeObject (consumer.p)

SmartMessageConsumer version of initializeObject.

Parameters: None

Returns: Nothing (procedure)

Notes: This redirects output to a log file when running in batch mode. If a JMS Session is started without errors, processDestinations is run to get the destinations needed for creating message consumers. When running in batch mode, startWaitFor is invoked to start the Adapter's waitForMessages.

initializeObject (containr.p)

Does container-specific initialization. If the container is a SmartFrame or nonvisual container, createObjects is run at this time. It is not run for a SmartFrame when it is first created to avoid creating objects that are not needed until after startup. If there is a StartPage (a first page to position to other than page 0), it is selected. If the HideOninit or DisableOnInit property is set for the Container, it is propagated to all contained objects before they are initialized. Then the container publishes "initializeObject" to initialize all its contents. Finally it does RUN SUPER to execute the initialization code in `smart.p`.

Parameters: None

Returns: Nothing (procedure)

Notes: None

initializeObject (data.p)

Performs SmartDataObject-specific initialization. Specifically, starts the server-side copy of this object if the AppService (partition) is defined.

Parameters: None

Returns: Nothing (procedure)

Notes: None

initializeObject (datavis.p)

Data visualization version of initializeObject. Performs initialization appropriate to visual objects that display data values. Part of the general initialization process. Sets certain property values and displays if any nondatabase-related fields.

Parameters: None

Returns: Nothing (procedure)

Notes:

- Invokes initializeObject in `smart.p` (RUN SUPER), first.
- Sets the value of EnabledHandles, CreateHandles, and FieldHandles property.
- Disables any associated Update SmartPanel or SmartToolbar it is connected to if no Update target is present.
- Invokes displayObjects to display nondatabase fields that might be present in the data visualization object.

initializeObject (field.p)

Special code for initializing SmartDataFields.

Parameters: None

Returns: Nothing (procedure)

Notes: This procedure stores the procedure handle of the SmartDataField in the PRIVATE-DATA of its Frame, in order that the procedure can be located from the frame. It also adds itself to the DisplayedFields property of its containing SmartDataViewer if the DisplayField logical property is TRUE, and, if EnableField is TRUE, to the EnabledFields property as well.

initializeObject (filter.p)

Filters object initialization code.

Parameters: None

Returns: Nothing (procedure)

Notes: Dynamic widgets are created for the list of fields specified in DisplayedFields. The deleteObjects() function is called to clean up to reflect changes in the Instance Property dialog. WordIndexedFields property is **never** checked again. Operator objects will assume that CONTAINS is allowed if the filter field object type is EDITOR. UseContains will also only work together with EDITOR fields. RANGE will **not** be used for word-indexed fields unless UseContains is FALSE.

initializeObject (messaging.p)

Initializes objects of class Messaging.

Parameters: None

Returns: Nothing (procedure)

Notes: Creates the JMS session and sets the Broker URL in the session. Also gets and sets the user, password, and clientID in the session before starting the session.

initializeObject (panel.p)

SmartPanel-specific initialization.

Parameters: None

Returns: Nothing (procedure)

Notes:

- A SmartPanel is set to the appropriate state depending on its Navigation–Target or Update–Target. A Navigation SmartPanel retrieves the QueryPosition property of its Navigation–Target to determine which buttons to enable. These QueryPosition values map to the value of the PanelState property that is set in the SmartPanel:
  - "FirstRecord" maps to "first".
  - "LastRecord" maps to "last".
  - "NotFirstOrLast" maps to "enable–all".
  - "NoRecordAvailable" maps to "disable–all".
- Likewise, an Update Panel retrieves the RecordState property of its Update–Target to do the same thing. These changes to the PanelState property are interpreted by the setButtons procedure in the SmartPanels.
  - "NoRecordAvailable" maps to "add–only".
  - "NoRecordAvailableExt" maps to "disable–all".
  - "RecordAvailable" maps to "initial".

initializeObject (producer.p)

Initializes objects of class Producer.

Parameters: None

Returns: Nothing (procedure)

Notes: None

initializeObject (query.p)

Initialization code for query objects.

Parameters: None

Returns: Nothing (procedure)

Notes: The query is not opened if it has a datasource. This is because a datasource means the object is dependent on data from another object. In this case the Data–Target (this procedure) waits until the dataAvailable event tells it to open its query. Instead, this procedure RUNs dataAvailable to see if its datasource is already running with a row available for it; in that case the dataAvailable will open the query.

initializeObject (router.p)

Initializes objects of class Router.

Parameters: None

Returns: Nothing (procedure)

Notes: None

initializeObject (sbo.p)

Initializes objects of class SBO. Sets the MasterDataObject property to the leading SmartDataObject.

Parameters: None

Returns: Nothing (procedure)

Notes: Later this procedure must also establish the AppServer connection and other tasks.

initializeObject (select.p)

Overrides initializeObject in order to subscribe to the ChangedEvent, if defined, and to initializeSelection.

Parameters: None

Returns: Nothing (procedure)

Notes: None

initializeObject (smart.p)

Performs general initialization common to all objects.

Parameters: None

Returns: Nothing (procedure)

Notes:

- There is a version of initializeObject in virtually every Super procedure; each performs the initialization appropriate to that class of objects. This top-level version runs the createControls and control_load procedures, if they exist, to initialize ActiveX Controls in the object, and sets the ObjectInitialized property to TRUE.
- Initialization of SmartObjects takes place in two phases. In the first phase, the constructObject procedure is run (normally from the AppBuilder-generated procedure adm–create–objects) for each SmartObject in a SmartWindow or other container. This runs the Progress persistent procedures which instantiates the object, and initializes Instance Properties for which values have been defined. adm–create–objects then creates links between objects. Once this is complete, the container will run initializeObject. This passes the initializeObject event down through all the contained objects. Thus any version of initializeObject can assume that all the SmartObjects and SmartLinks in a container have been established; thus it can check, for example, whether there is a link to some other particular kind of object. Code should not, however, assume the order in which SmartObjects will be initialized. For example, a SmartDataObject will open its query and publish “dataAvailable” to signal that to other objects. A SmartDataViewer will run dataAvailable in itself to see if there is already a row waiting for display, which would happen if the associated SmartDataObject was initialized first.

- Application code can therefore localize createObjects in a SmartContainer to add code to the creation phase (for example, to define additional application-specific links or set application-specific properties that need to be looked at during initialization), or to initializeObject to add code to the initialization phase after all objects have been created and all links created. For noncontainer SmartObjects, code to be executed during the creation phase (that is, when the object's procedure is first run) should be placed in the Main Block. Code to be executed after the object and other related objects have been created should be placed into a local initializeObject.

initializeObject (toolbar.p)

Initializes the toolbar by creating all dynamic buttons and menus.

Parameters: None

Returns: Nothing (procedure)

Notes: None

initializeObject (tvcontroller.p)

TreeViewController initialization procedure.

Parameters:

Returns: Nothing (procedure).

Notes: None.

initializeObject (viewer.p)

Enables fields in the list represented by the AppBuilder generated preprocessor value {&ENABLED–FIELDS}.

Parameters: None

Returns: Nothing (procedure)

Notes:

- The ENABLED–FIELDS preprocessor value contains a list of RowObject fields that are enabled in a SmartDataViewer. At object initialization, this is turned into the EnabledHandles property, which holds a list of the widget handles of the enabled fields in the SmartDataViewer's frame, for efficient access.

- If no TableIO–Source (or GroupAssign–Source connected to a TableIO–Source for SmartDataViewers) is present or no record is available, the fields are not enabled.
- enableFields republishes the “enableFields” event to handle enabling fields in any GroupAssign targets.
- enableFields can be customized when you require additional processing when RowObject fields are enabled. You might want to do this to enable or hide other widgets not associated with RowObject fields.

initializeObject (visual.p)

Initialization code specific to visualizations.

Parameters: None

Returns: Nothing (procedure)

Notes:

- Invoked with RUN SUPER from `containr.p`’s version of initializeObject.
- Reads through the list of enabled objects (which does not include RowObject fields) and gets their handles to store in EnabledObjHdls property.
- Invokes enableObject and viewObject depending on the values of DisableOnInit and HideOnInit property settings.
- Invokes applyLayout to change the object to the correct layout if there are multiple layouts.

initializeSelection (select.p)

Unless the view-as is specified to be a browse, creates a selection widget and populates it with data.

Parameters: None

Returns: Nothing (procedure)

Notes: None

initializeServerObject (appserver.p)

Initializes the server object after it has been started or restarted.

Parameters: None

Returns: Nothing (procedure)

Notes: This is an internal “event” that is called from runServerObject after a successful run on server. It silently ignores any calls when not bound.

initializeServerObject (data.p)

Initializes and sets context in the serverObject after it has been started or restarted.

Parameters: None

Returns: Nothing (procedure)

Notes: None

initializeServerObject (sbo.p)

Sets context and initializes the server object.

Parameters: None

Returns: Nothing (procedure)

Notes: None

initializeVisualContainer (containr.p)

Translates window title and tab folder page labels and checks security for page labels.

Parameters: None

Returns: Nothing (procedure)

Notes: None

initPages (containr.p)

Initializes one or more pages, which are not yet being viewed, in order to establish links or to prepare for the pages being viewed.

Parameters:

```
INPUT pcPageList AS CHARACTER
```

A comma-separated list of page numbers.

Returns: Nothing (procedure)

Notes:

- The initPages procedure can be run by application code to initialize more pages than those that are initially viewed, either in order to reduce the startup time for pages later on, or to communicate with objects on pages not yet selected.
- If a SmartObject on one page in a SmartContainer is a link target of a SmartObject on another page other than page 0, the AppBuilder will detect this during the design process and will automatically generate an appropriate call to initPages as part of adm–create–objects, to assure that when the dependent page is initialized, the pages it depends on will be initialized as well.
- Ordinarily only objects on page 0 and on the StartPage (if any) are created when the SmartContainer is created. Objects on other pages are created when the page they are on is first selected.

- In some cases a SmartDataObject can be an Update Target for a SmartDataViewer or Browser, which can be on a later page in the Container. This might cause more pages to be created at startup than is desired. If this is the case, the design time Update link can be removed, and created instead by application code in a local createObjects, as in the second example below.

Examples:

```
PROCEDURE initializeObject:

/* Purpose: Initialize the current SmartContainer and initialize some
   frequently used pages at the same time. */

  RUN SUPER.   /* Execute standard page creation first. */
   /* Initialize pages 2, 8, and 106 at startup. This will create the
      pages but not view them. */
  RUN initPages ('2,8,106':U).
END PROCEDURE.

PROCEDURE createObjects:

/* Purpose: Local version to add an Update link between a SmartDataObject
   on page 0 and a SmartDataViewer on page 2; not done until page 2 is
   created. */

  RUN SUPER.       /* Do the standard creation for each page first. */
   IF getCurrentPage() = 2 THEN
     RUN addLink(h_Vcust, 'Update':U, h_Dcust).
END PROCEDURE.
```

initProps (query.i)

During initialization, sets all the object's properties that are related to the Query.

Parameters: None

Returns: Nothing (procedure)

Notes: None

insertExpression (combo.p)

Inserts an expression into ONE buffer’s WHERE clause.

Parameters:

`INPUT pcWhere AS CHARACTER`

A complete WHERE clause with or without the FOR keyword, and without leading or trailing commas.

`INPUT pcExpression AS CHARACTER`

New expression/OF phrase to replace the existing OF phrase.

`INPUT pcAndOr AS CHARACTER`

Whether to AND (default) or OR the new expression.

Returns: CHARACTER PRIVATE

Notes:

- The new expression is embedded in parentheses, but no parentheses are placed around the existing expression.
- Lock keywords must be unabbreviated or without –LOCK (that is. SHARE or EXCLUSIVE.)
- Any keyword in comments might cause problems.

insertExpression (lookup.p)

PRIVATE function. Inserts an expression into one buffer's WHERE clause.

Parameters:

`INPUT pcWhere AS CHARACTER`

A complete WHERE clause with or without the FOR keyword, and without leading or trailing commas.

`INPUT pcExpression AS CHARACTER`

New expression/OF phrase to replace the existing OF phrase.

`INPUT pcAndOr AS CHARACTER`

Whether to AND (default) or OR the new expression.

Returns: CHARACTER PRIVATE

Notes:

- The new expression is embedded in parentheses, but no parentheses are placed around the existing expression.
- Lock keywords must be unabbreviated or without –LOCK (that is, SHARE or EXCLUSIVE.)
- Any keyword in comments might cause problems.

insertExpression (query.p)

Inserts an expression into a buffer's WHERE clause.

Parameters:

`INPUT pcWhere AS CHARACTER`

Complete WHERE clause with or without the FOR keyword, but without commas.

`INPUT pcExpression AS CHARACTER`

New expression **or** OF phrase (Existing OF phrase is replaced).

`INPUT pcAndOr AS CHARACTER`

Specifies what operator is used to add the new expression to existing ones, either AND (the default) or OR.

Returns: CHARACTER PRIVATE

Notes:

- The new expression is embedded in parentheses, but no parentheses are placed around the existing expression.
- Lock keywords must be unabbreviated or without –LOCK (for example, SHARE or EXCLUSIVE).
- Any keyword in comments might cause problems.
- This is PRIVATE to `query.p`.

insertFieldProperty (filter.p)

Inserts or adds a fields property to the CHR(1)-separated internal property that holds fields and values.

Parameters:

`INPUT pcList AS CHARACTER`

CHR(1)-separated list with fieldnames and values.

`INPUT pcFields AS CHARACTER`

Fieldname.

`INPUT pcValue AS CHARACTER`

Value of the field property.

Returns: CHARACTER PRIVATE

Notes: None

insertMenu (toolbar.p)

Creates a set of menu items or submenus under a parent menu.

Parameters:

`INPUT pcParent AS CHARACTER`

The unique action name of an already-inserted parent. Blank means this is the menu bar itself.

`INPUT pcActions AS CHARACTER`

Comma-separated list of actions or ActionGroups. The term RULE specifies that delimiter.

`INPUT plExpand AS LOGICAL`

If TRUE, all actions in the action group are added under this parent. If FALSE, action groups are created as submenus.

`INPUT pcBefore AS CHARACTER`

The unique action name of an already-inserted sibling.

Returns: LOGICAL

Notes: None

instancePropertyList (smart.p)

Returns a list of the values of the names of the object's InstanceProperties, that is, those properties that can be set to initial values in design mode. These can be set in the AppBuilder to determine the object instance's behavior at run time.

Parameters:

`INPUT pcPropList AS CHARACTER`

Optional list of properties wanted. If this parameter is blank, the default is all of the instance properties. Other valid options are "*"(all properties), or a list of the specific properties wanted.

Returns: CHARACTER (Specially delimited list of property names and values.)

Notes:

- The properties are returned in a string delimited by CHR(3) between property name/value pairs, and CHR(4) between the name and the value.
- Normally used internally by the AppBuilder at application design time to retrieve a list of property names and values to insert into the generated code in the adm–create–objects procedure.
- If the input parameter is the special value “ADM–TRANSLATABLE–FORMAT”, then the property list string is returned in precisely the form used in generating adm–create–objects, with the special translation suffix “:U” inserted in the list following nontranslatable property values, and the suffix left off for special translatable properties such as the Tab Folder’s FolderLabels property.
- If the input parameter is the special value "*", then all object properties are returned with their values. A list of object properties is determined by identifying all the *get* property functions for the object (functions beginning "get" with no following hyphen, taking no input parameters), plus all dynamic properties stored in the UserProperty string.
- If the input parameter is a comma-separated list of property names, just those properties and their values are returned.

internalSchemaFile (router.p)

Returns the internal Schema filename based on the external and internal file reference definitions.

Parameters:

```
INPUT   pcNameSpace AS CHAR
```

Target Namespace, the XMLNS attribute from the incoming XML document.

Returns: CHARACTER (filename)

Notes: None

isNodeExpanded (treeview.p)

Returns TRUE if some node is expanded.

Parameters:

```
INPUT pcNodeKey AS CHARACTER
```

The node in question

Returns: LOGICAL

Notes: None.

isObjQuoted (smart.p)

Returns TRUE if the object number in the query string is represented as a quoted (sub)string. European-format (999,99) numbers must not be reformatted (999.99) if represented as a string.

Parameters:

```
INPUT pcQueryString AS CHARACTER
```

The query string to be examined.

```
INPUT piPosition AS INTEGER
```

The index into the string

Returns: LOGICAL

Notes: This is needed when running the application using European format.

isUpdateActive (containr.p)

Received from container source to check if contained objects have unsaved or uncommitted changes (including addMode).

Parameters:

```
INPUT-OUTPUT plActive AS LOGICAL
```

Returns: Nothing (procedure).

Notes: This is published thru the container link and used mainly to validate that a false value received in updateActive can be used to set UpdateActive. This is NOT intended to be called directly, but part of the logic that updates UpdateActive. These are the steps: (1) Updating objects publishes updateActive (true or false) to their container targets. (2) If the value is FALSE the container then publishes this to ALL ContainerTargets before it is stored in UpdateActive. This way the value is only stored as FALSE if ALL contained objects are inactive.

isUpdateActive (data.p)

Received as a publication from container source to check whether contained objects have unsaved or uncommitted changes, including addMode.

Parameters:

```
INPUT-OUTPUT plActive AS LOGICAL
```

FALSE if the ContainerTargets should be surveyed.

Returns: Nothing (procedure).

Notes:

- This is published thru the container link from canExit for close logic (ok, cancel, exit). It is very similar to canNavigate -> isUpdatePending, which is published thru the data link.
- We currently ONLY check rowObjectState as the other states are checked in the visual objects.

isUpdateActive (datavis.p)

Received from container source to check whether contained objects have unsaved or uncommitted changes, including addMode.

Parameters:

```
INPUT-OUTPUT plActive AS LOGICAL
```

Returns: Nothing (procedure).

Notes: This is published thru the container link and is used for close logic (ok, cancel, exit). It is very similar to canNavigate->isUpdatePending which is published thru the data link. This can be called directly, but canExit() is the intended API. canExit() is used to check an actual container from the toolbar.

isUpdatePending (data.p)

If no updates are known to be pending (RowObjectState is not RowUpdated, neither NewRow nor DataModified are TRUE), publishes isUpdatePending to data-targets to check. Otherwise returns without action.

Parameters:

```
INPUT-OUTPUT plUpdate AS LOGICAL
```

False if this routine should check for pending updates.

Returns: Nothing (procedure)

Notes: New is included as a pending update. Called from canNavigate, which is used by navigating objects to check if they can trust an updateState("updatecomplete") message. This check is only valid from a DataSource point of view. Use canNavigate to check an actual object.

isUpdatePending (sbo.p)

Published through data-targets to check if any updates are pending. This SBO version of the event turns around and RUNS it in the SDO to which the caller is mapped. If no pending updates are found, it publishes isUpdatePending to its targets.

Parameters:

```
INPUT-OUTPUT plUpdate AS LOGICAL
```

Returns TRUE and stops the publication if update is pending.

Returns: Nothing (procedure)

Notes:

- New is included as a pending update.
- This routine is called from canNavigate, which is used by navigating objects to check whether they can trust an updateState("UpdateComplete") message.

launchFolderWindow (browser.p)

Calls launchContainer in the session manager to start a folder window, if there is not already one available.

Parameters:

```
INPUT pcContainerMode AS CHARACTER
```

Returns: Nothing (procedure).

Notes: None.

leaveCombo (combo.p)

Trigger fired on leave of combo field. It stores the current screen value in the combo.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

leaveLookup (lookup.p)

This procedure is fired on leave of a lookup field.

Parameters: This procedure handles the situation where a value is manually entered rather than using the lookup key. When that happens, we need to see whether the screen value has been changed. If it has, see if the changed screen value is a valid record. If it is, reset the keyvalue thus avoiding having to use the lookup.

Returns: Nothing (procedure).

Notes:

leaveSelect (select.p)

This procedure, fired on leave of a SmartSelect field, determines whether the lookup can be avoided because the new value exists as a unique record.

Parameters:

Returns: Nothing (procedure).

Notes: This works only if the displayed field is in the first buffer of the linked SDO.

linkHandles (design.p)

Returns a comma-separated list of linked objects at design time

Parameters:

```
INPUT pcLink AS CHARACTER
```

The link of interest. Example: 'DATA-SOURCE'.

Returns: CHARACTER

Notes: None.

linkHandles (smart.p)

Takes a link name and returns a list of handles of objects at the other end of that link, relative to the TARGET–PROCEDURE.

Parameters:

```
INPUT pcLink AS CHARACTER
```

The link name (including “-SOURCE” or “-TARGET”).

Returns: CHARACTER (Comma-separated list of handles to the SmartObject procedures at the other end of the link.)

Notes:

- This is the Version 9 (ADM2) equivalent of the procedure get–link–handle in Version 8.
- If the link type does not exist in the object, then the empty string ("") is returned.
- The procedure handle list is returned as a character string because (if the link type requested supports multiple objects at that end of the link) there might be multiple SmartObjects connected by that link type, so these come back as a comma-separated list. The developer must check the NUM–ENTRIES in the returned list (which might be zero, one, or more), and apply the WIDGET–HANDLE function to each entry to derive the procedure’s handle.

Examples:

```
/*
** Here, a pass-through link is created from a SmartDataObject in a parent
** SmartWindow, through the SmartWindow containing this code, to a
** SmartDataBrowser contained in this window. The Data link from the SDO will
** go both to the SmartWindow and to the Browser. There is no standard
** implementation of the 'dataAvailable' event that is published whenever a
** new row is selected in the parent SDO, but this local version will intercept
** that event and use it to modify the title of the window.
*/

PROCEDURE dataAvailable:
  /* This input parameter is defined but we don't look at it. */
  DEFINE INPUT PARAMETER cType AS CHARACTER NO-UNDO.
  DEFINE VARIABLE hDataSource AS HANDLE NO-UNDO.
  DEFINE VARIABLE cValues     AS CHAR   NO-UNDO.
  DEFINE VARIABLE hWindow     AS HANDLE NO-UNDO.
  /* We can convert the return value directly to a handle because we know
     there is only one
Data Source. */
hDataSource = WIDGET-HANDLE(dynamic-function('linkHandles':U,
'Data-Source':U)).
  /* Ask for the Customer Name field from a Customer SDO. */
   cValues = dynamic-function('colValues':U IN hDataSource, INPUT
    'Name':U).
   /* Get the widget handle of the window itself. */
   hWindow = dynamic-function('getContainerHandle':U).
   /* First value returned from colValues is always the RowIdent, so
      skip it. */
  hWindow:TITLE = "Orders for Customer " + ENTRY(2, cValues, CHR(1)).
END PROCEDURE.
```

linkProperty (smart.p)

Returns the value of the requested property in the object at the other end of the specified link, relative to TARGET–PROCEDURE.

Parameters:

`INPUT pcLink AS CHARACTER`

The link name.

`INPUT pcPropName AS CHARACTER`

The property name.

Returns: CHARACTER: property value in string form.

Notes:

- This function is the Version 9 (ADM2) equivalent of the request–attribute procedure in Version 8.
- The value is returned in character format, regardless of its native datatype. If there is not exactly one object at the other end of the link, or that object is no longer there, the unknown value is returned.

Examples:

```
/* This example determines whether the AutoCommit property is true in the
   SmartdataObject associated with the current visual object. */

lCommit = dynamic-function('linkProperty':U, INPUT 'Data-Source':U, INPUT
   'AutoCommit':U).
```

linkState (panel.p)

Receives a message whenever a linked object becomes active (typically when it is viewed) or inactive (hidden).

Parameters:

```
INPUT pcState AS CHARACTER
```

Allowed values: ACTIVE and INACTIVE.

Returns: Nothing (procedure)

Notes: None

linkState (toolbar.p)

Receives messages when an object linked to this one becomes active (typically when it is viewed) or inactive (hidden). Resets buttons accordingly.

Parameters:

```
INPUT pcState AS CHARACTER
```

Valid values: ACTIVE, INACTIVE.

Returns: None (procedure)

Notes: None

loadActions (action.p)

Calls getActions in the Repository Manager to load the actions listed in the argument.

Parameters: None.

```
INPUT pcActionList AS CHARACTER
```

The list of actions to be loaded into ttAction and ttCategory

Returns: Nothing (procedure).

Notes: None.

loadBands (action.p)

Calls getToolbarBandActions in the Repository Manager to load data for one or more bands.

Parameters:

```
INPUT pcBandList AS CHARACTER
```

The list of bands to be loaded

```
OUTPUT TABLE-HANDLE hBand


OUTPUT TABLE-HANDLE hBandAction
```

Returns: Nothing (procedure).

Notes: None.

loadCategories (action.p)

Load specified actions.

Parameters:

```
INPUT pcCategoryList AS CHARACTER
```

The list of actions to be loaded

Returns: Nothing (procedure).

Notes: None.

loadImage (treeview.p)

Returns the index of some image file in the ImageList, first loading it if necessary. If the load fails, the function returns 0 (zero).

Parameters:

```
INPUT pcImageFile AS CHARACTER
```

The file of interest

Returns: INTEGER PRIVATE

Notes: The filename must specify the relative path and filename with the extension. Since the imagelist does not contain any images, the ImageList property is assigned to the Treeview only once, after an image is added.

loadObjectBands (action.p)

Calls getToolbarBandActions to load bands and actions for one or more objects

Parameters:

```
INPUT pcObjectList AS CHARACTER


OUTPUT TABLE-HANDLE hObjectBand


OUTPUT TABLE-HANDLE hBand


OUTPUT TABLE-HANDLE hBandAction
```

Returns: Nothing (procedure).

Notes: None.

loadPanel (panel.p)

Loads actions.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

loadProducerSchema (b2b.p)

Returns the result of identifying and loading the appropriate producer schema.

Parameters: None

Returns: LOGICAL

Notes: None

loadSchema (b2b.p)

Loads the schema temp-table.

Parameters:

```
INPUT pcSchema AS CHARACTER
```

Returns: LOGICAL

Notes: None

loadToolbar (toolbar.p)

Loads toolbar, bands and actions for the toolbar object.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

loadToolbarBands (action.p)

Load bands and actions for a toolbar

Parameters:

`INPUT pcToolbar AS CHARACTER`

The toolbar of interest

`INPUT pcObjectList AS CHARACTER`

`OUTPUT TABLE-HANDLE hToolbarBand`

`OUTPUT TABLE-HANDLE hObjectBand`

`OUTPUT TABLE-HANDLE hBand`

`OUTPUT TABLE-HANDLE hBandAction`

Returns: Nothing (procedure).

Notes: None.

mapNode (b2b.p)

Maps a node to a procedure, function, or column in a data object.

Parameters:

```
INPUT pdNode AS DECIMAL

INPUT phDataSource AS HANDLE

INPUT pcMapType AS CHARACTER

INPUT pcMapName AS CHARACTER

INPUT pcConversion AS CHARACTER

INPUT pcMapParameter AS CHARACTER

INPUT pcNodeType AS CHARACTER

INPUT pcNodeName AS CHARACTER
```

Returns: DECIMAL

Notes: If the node is mapped to more than one parameter of the Data-Source method, the mapping is stored in storeParameters and the data will be processed in callOutParams().

mappedEntry (smart.p)

Returns the other entry in a separated list of paired entries. This is required to ensure that the lookup does not find a matching entry in the wrong part of the pair.

Parameters:

`INPUT pcEntry AS CHARACTER`

Entry to lookup.

`INPUT pcList AS CHARACTER`

Comma-separated list with paired entries.

`INPUT plFirst AS LOGICAL`

If TRUE, lookup first and return second. If FALSE, lookup second and return first.

`INPUT pcDelimiter AS CHARACTER`

Delimiter of pcList.

Returns: CHARACTER

Notes: Used to find mapped RowObject or database column in assignList. In other cases, such as the ObjectMapping property of SBOs, an entry might occur more than once in the list, in which case a list of matching values is returned, using the same delimiter as the list.

menuHandle (toolbar.p)

Returns the handle of a menu.

Parameters:

`INPUT pcName AS CHARACTER`

The action name of a menu.

Returns: HANDLE

Notes: PRIVATE function.

messageHandler (consumer.p)

Runs when a message is received. It handles the incoming message by calling appropriate procedures in its INMESSAGE-TARGET.

Parameters:

```
INPUT phMessage AS HANDLE

INPUT phMessageConsumer AS HANDLE

OUTPUT phReply AS HANDLE
```

Returns: Nothing (procedure)

Notes: None

messageNumber (smart.p)

Returns the message text given a message number. Allows these messages to be translated and tracked in one place.

Parameters:

```
INPUT piMessage AS INTEGER
```

Returns: CHARACTER

Notes:

- In order to allow certain messages to be translated, it is helpful to group them into a single file, since the ADM super procedure source files are not otherwise regarded as part of a translatable application. To facilitate this, messages that are expected to be seen by end users and, therefore, should be translated are located in a single include file (`src/adm2/admmsgs.i`) in the form of a character array. This allows a translated version of this single file to be substituted and `smart.p`, which includes `admmsgs.i`, to be recompiled or run through the Translation Manager tool in order to translate end-user messages that are raised from code located in super procedures.
- The messageNumber function is normally invoked from the addMessage procedure, as in the example.
- The ADM convention is that error messages that are expected to be seen only by developers or application testers (errors indicating errors in application construction, for example, caused by missing links or the like) are not translated, and, therefore, appear in the super procedures as literal strings with the ":U" suffix.

Examples:

```
/* This example retrieves the text for message number 4 ("Current values
   must be saved or cancelled before Commit.") and passes it to the
   addMessage procedure to add that message to the error message log. */

RUN addMessage (messageNumber(4), ?, ?).
```

modifyDisabledActions (toolbar.p)

Modifies the DisabledActions property and makes it possible to permanently disable actions independent of state changes.

Parameters:

`INPUT pcMode AS CHARACTER`

If "ADD", adds the actions to the DisabledActions. If "REMOVE", removes the actions from the DisabledActions.

`INPUT pcActions AS CHARACTER`

Comma-separated list of actions.

Returns: LOGICAL

Notes: If pcMode = "ADD", the actions will be immediately disabled and subsequent calls to enableActions will not enable them again. If pcMode = "REMOVE", the actions that are removed from the list will be enabled the next time they are called with enableActions.

modifyListProperty (smart.p)

Allows values to be added to or deleted from any object property that is a comma-separated list.

Parameters:

`INPUT phCaller AS HANDLE`

Handle of the object whose property is being changed.

`INPUT pcMode AS CHARACTER`

"ADD" or "REMOVE".

`INPUT pcListName AS CHARACTER`

The name of the property.

`INPUT pcListvalue AS CHARACTER`

The value to add or remove.

Returns: Nothing (procedure)

Notes:

- This is the ADM 2 equivalent of what was modify–list–attribute in the Version 8 ADM.
- Normally the first argument will be the handle THIS–PROCEDURE, if the property value is to be changed for the current SmartObject. However, this can be another procedure handle if the property is to be modified in another object.
- The modifyListProperty procedure first runs the get*propname* function to retrieve the current value of the property. If a new value is being added, and is already contained in the list, or if a value to be removed is not present in the list, modifyListProperty simply returns without error. Otherwise, the change to the list is made and the set*propname* function is run to reset the value of the list property. Both the *get* and *set* functions must exist; otherwise, modifyListProperty will return without error.
- There are many ADM Properties that are expressed as comma-separated lists of handles or other values. All of these should be maintained using the modifyListProperty procedure. Using the set*propname* function will reset the entire list to the value, which is normally not what is desired.

Examples:

```
/* This code will add an additional entry to the SupportedLinks property
   for the object.*/

RUN modifyListProperty (INPUT THIS-PROCEDURE, INPUT 'ADD':U, INPUT
   'SupportedLinks':U, INPUT 'SpecialLink':U).

/* This code adds the specified source procedure handle to the list of
   "Source"s for the specified link in the specifed Target procedure
    handle (from addLink). */

RUN modifyListProperty (INPUT phTarget, INPUT 'ADD':U, INPUT
  pcLink + "Source":U, INPUT STRING(phSource)).
```

modifyUserLinks (smart.p)

Maintains a delimited list of user-defined links (that is, links that are not in the SupportedLinks list for an object), and the handles of the objects at the other end of the links.

Parameters:

`INPUT pcMode AS CHARACTER`

"ADD" or "REMOVE".

`INPUT pcLinkName AS CHARACTER`

The link name including -Source or -Target.

`INPUT phObject AS HANDLE`

The procedure handle of the object at the other end of the link.

Returns: Nothing (procedure)

Notes:

- Run from addLink and removeLink; used primarily by the linkHandles function. When addLink encounters a link that is not in the list of SupportedLinks, it defines a single SUBSCRIBE for an event of that name (see addLink). Because there are no properties where the objects at either end of a nonsupported (or “dynamic”) link can be stored, they are stored in this list so that functions such as linkHandles, which returns the handle of an object at the other end of a link, can keep track of what the relationships are.
- This function is not intended to be run from application code.
- The list is the third entry in ADM–DATA, delimited by CHR(1). Each entry in the list consists of a link name followed by CHR(4) followed by a comma-separated list of one or more handles. The list entries are delimited by CHR(3). Users should not be concerned about the specific format and location of the list, which might be subject to change; using this procedure to access the list will preserve compatibility with any possible changes.

movebuttons (toolbar.p)

Calculates start positions for left, right, and centre aligned bands in the toolbar according to the current width of the toolbar frame, then move the existing buttons into their new positions. This is called from resizeObject so it is always current for the actual size of the toolbar frame.

Parameters:

```
INPUT pcForceMove AS LOGICAL
```

YES if the buttons should be forced to move

Returns: Nothing (procedure).

Notes: This routine expects already-built buttons. The first thing done is see if there are any right-aligned or center-aligned bands and if not, we can just exist as there is nothing to do. Also if all buttons are left aligned there is nothing to do. We must ignore hidden buttons. When called from viewHideActions procedure after buttons have been viewed/hidden, then we must force this procedure to run in order to correctly position buttons and not leave any gaps.

moveMenu (toolbar.p)

Moves and re-parents some child menu. Creates a new tree and deletes the existing tree so make sure the next-sibling is available before this is called if this is done in a widget-tree loop (or use moveMenuChildren).

Parameters:

```
INPUT phMenu AS HANDLE
```

The menu to move

```
INPUT phNewParent AS HANDLE
```

The new parent

Returns: HANDLE

Notes: None.

moveMenuChildren (toolbar.p)

Moves and re-parents some group of child menus by calling moveMenu in a loop.

Parameters:

`INPUT phOldParent AS HANDLE`

The parent losing the children

`INPUT phNewParent AS HANDLE`

The parent gaining the children

Returns: LOGICAL

Notes: None.

newDataObjectRow (sbo.p)

Tells one or more contained SDOs to prepare to create (add or copy) a new record.

Parameters:

`INPUT pcMode AS CHARACTER`

The operation to be performed. Valid values are "Add" and "Copy".

`INPUT pcTargetNames AS CHARACTER`

The list of SDOs to be notified. These might be qualified with the ObjectName.

`INPUT pcViewColList AS CHARACTER`

The list of column names.

Returns: CHARACTER

Notes: Called by addRow and copyRow.

newQueryString (combo.p)

Returns a new query string to the passed query. The tables in the passed query must match getQueryTables(). Adds column / value pairs to the corresponding buffer's where-clause. Each buffer's expression will always be embedded in parenthesis.

Parameters:

`INPUT pcColumns AS CHARACTER`

Comma-separated list of column names of the form table.fieldname

`INPUT pcValues AS CHARACTER`

Chr(1)-separated list of corresponding values

`INPUT pcDataTypes AS CHARACTER`

Comma-separated list of corresponding data types

`INPUT pcOperators AS CHARACTER`

Comma-separated list of corresponding operators. If blank, the operator is EQ for all columns. Alternative string operators can be separated with a slash, for example: EQ/BEGINS.

`INPUT pcQueryString AS CHARACTER`

A complete query string. This must be qualified correctly and match the queried tables. If "?", the base query is presumed.

`INPUT pcAndOr AS CHARACTER`

The operator that defines how the new clause is appended, for each buffer, to the query in pcQueryString. Valid values are "AND" and "OR".

Returns: CHARACTER

Notes: This was taken from query.p but changed for Combos to work without an SDO.

newQueryString (lookup.p)

Returns a new query string to the passed query. The tables in the passed query must match getQueryTables(). Adds column/value pairs to the corresponding buffer's WHERE clause. Each buffer's expression will always be enclosed in parentheses.

Parameters:

`INPUT pcColumns AS CHARACTER`

Comma-separated list of column names, of the form table.fieldname

`INPUT pcValues AS CHARACTER`

Comma-separated list of the corresponding values

`INPUT pcDataTypes AS CHARACTER`

Comma-separated list of the corresponding datatypes

`INPUT pcOperators AS CHARACTER`

Comma-separated list of operators. If a single operator, applies to all columns. If blank, defaults to EQ. Separate alternative string operators with a slash; example: "EQ/BEGINS"

`INPUT pcQueryString AS CHARACTER`

A complete, fully-qualified query string matching the queries tables. If "?", use the base query.

`INPUT pcAndOr AS CHARACTER`

How to relate the new expression to the existing query (for each buffer). Valid values are "AND" and "OR".

Returns: CHARACTER

Notes: This was taken from query.p but changed for lookups to work without an SDO.

newQueryString (query.p)

Returns a new query string to the passed query. The tables in the passed query must match getTables(). Adds column/value pairs to the corresponding buffer's WHERE clause. Each buffer's expression will always be embedded in parentheses.

Parameters:

`INPUT pcColumns AS CHARACTER`

Comma-separated column names of a table in the query in the form of TBL.FLDNM or DB.TBL.FLDNM. RowObject.FLDNM should be used for SDOs. If the fieldname is not qualified, it checks the tables in the TABLES property and assumes the first with a match.

`INPUT pcValues AS CHARACTER`

Corresponding values (CHR(1)-separated).

`INPUT pcOperators AS CHARACTER`

Operator for all columns. Blank defaults to EQ. Use a slash to define an alternative string operator. Comma-separated for each column/value.

`INPUT pcQueryString AS CHARACTER`

A qualified query string matching the queries tables.

`INPUT pcAndOr AS CHARACTER`

AND or OR decides how the new expression is appended to the passed query (for each buffer).

Returns: CHARACTER

Notes: None

newQueryValidate (query.p)

Inserts a new expression to the passed prepare string if the buffer reference, usually extracted from the expression, is valid. Returns ? if error (after an error message has been shown).

Parameters:

`INPUT pcQueryString AS CHARACTER`

Complete query string that the expression is going to be added to.

`INPUT pcExpression AS CHARACTER`

The new expression to add to the query. The first field reference is used as buffer reference unless pcBuffer is defined.

`INPUT pcBuffer AS CHARACTER`

Optional buffer reference.

`INPUT pcAndOr AS CHARACTER`

Operator used to append if query string already has an expression. Default is "=".

Returns: CHARACTER

Notes:

The function avoids duplicate logic in add and setQueryWhere and acts as a wrapper for newWhereClause with some significant additions and differences:

- The buffer reference is usually extracted from the new expression.
- pcBuffer is optional and thus not the first parameter.
- The buffer reference is validated and qualification does not need to match the original query.
- Displays error message and returns "?" if the buffer is unknown or ambiguous.

newQueryWhere (query.p)

Returns a new query.

Parameters:

```
INPUT pcWhere AS CHARACTER
```

Returns: CHARACTER

Notes: For internal use only. This functions exists only to have the same logic in setQueryWhere in `data.p` and `query.p`. Returns unknown if the buffer reference is invalid.

newRowObject (data.p)

Assigns some general RowObject fields and updates LastRowNum and FirstRowNum when a new RowObject has been created.

Parameters:

```
INPUT pcMode AS CHARACTER
```

The operation to be performed. Valid values are "Add" and "Copy".

Returns: LOGICAL

Notes:

- The main purpose for this procedure is to ensure that copy and add behaves similarly.
- The buffer must be created first.
- Currently defined as PRIVATE.
- Used in procedure copyColumns and function addRow().

newWhereClause (combo.p)

Inserts a new expression to query’s prepare string for a specified buffer.

Parameters:

`INPUT pcBuffer AS CHARACTER`

The buffer

`INPUT pcExpression AS CHARACTER`

The new expression

`INPUT pcWhere AS CHARACTER`

The current query-prepare string

`INPUT pcAndOr AS CHARACTER`

The operator used to relate the new expression to the existing query. Valid values are "AND" (default) and "OR".

Returns: CHARACTER

Notes: This is a utility function that doesn’t use any properties.

newWhereClause (lookup.p)

Appends a new expression to the query's prepare string for a specified buffer.

Parameters:

`INPUT pcBuffer AS CHARACTER`

The target buffer

`INPUT pcExpression AS CHARACTER`

The new expression

`INPUT pcWhere AS CHARACTER`

The current query-prepare string

`INPUT pcAndOr AS CHARACTER`

The operator with which to append the new expression: 'AND' (default) or 'OR'

Returns: CHARACTER

Notes: This is a utility function that doesn't use any properties.

newWhereClause (query.p)

Inserts a new expression to a specified buffer's WHERE clause.

Parameters:

`INPUT pcBuffer AS CHARACTER`

Buffer.

`INPUT pcExpression AS CHARACTER`

The new expression.

`INPUT pcWhere AS CHARACTER`

The current query prepare string.

```
INPUT pcAndOr AS CHARACTER
```

Specifies what operator is used to add the new expression to existing expressions, either AND (the default) or OR.

Returns: CHARACTER

Notes: This is supported as a utility function that does not use any properties. However, if target-procedure = super, the passed buffer's qualification must match the query's. If target-procedure <> super, the buffer will be corrected if it exists in the object's query. Otherwise, it needs to match.

nodeHandle (xml.p)

Returns the handle of the node identified by the `pdId` argument.

Parameters:

```
INPUT pdId AS DECIMAL
```

Returns: CHARACTER

Notes: None

nodeType (xml.p)

Returns the type of the node identified by *`pdId`*.

Parameters:

```
INPUT pdId AS DECIMAL
```

Returns: CHARACTER

Notes: None

normalizeActionData (action.p)

Removes duplicate entries from temp-tables after loading some set of actions.

Parameters: None.

Returns: LOGICAL

Notes: The procedure handle is used to identify data to be kept or purged

NotFoundError (b2b.p)

Returns a useful ‘Record not found’ error message with keyfields.

Parameters:

`INPUT phDataSource AS HANDLE`

DataSource

`INPUT pcKeyValues AS CHARACTER`

CHR(1) list with keyvalues corresponding to KeyFields

Returns: CHARACTER

Notes: None.

notifyPage (containr.p)

Invokes the specified procedure in all objects on the CurrentPage of A SmartContainer.

Parameters:

`INPUT pcProc AS CHARACTER`

The internal procedure to run.

Returns: Nothing (procedure)

Notes:

- This procedure is necessary because the notion of paging does not fit well with PUBLISH/SUBSCRIBE. All objects in a Container will, for example, subscribe to initialize (and so on), but the paging performs the operation on subsets of those objects.
- The notifyPage procedure will run an event procedure in all the objects on the current page, determined from the PageNtargets property.

Examples:

```
/* This code fragment from changePage causes initializeObject to be run
   in each newly created object on a page selected for the first time.
   NotifyPage effectively does a PUBLISH to each object on the current
   page. */

{get ObjectInitialized lInitted}. /* Is the container itself initialized? */
IF lInitted THEN
   RUN notifyPage In TARGET-PROCEDURE ("initializeObject":U).
```

numParameters (b2b.p)

Returns the number of parameters for a method.

Parameters:

`INPUT phProc AS HANDLE`

Valid procedure handle.

`INPUT pcMethod AS CHARACTER`

Function or procedure name.

Returns: INTEGER

Notes: None

obtainInMsgTarget (router.p)

Gets the handle of the incoming message and starts some container (containing, for example, a SmartB2BObject and a business object such as a SmartDataObject). It then returns the handle of the INMESSAGE-TARGET from that container.

Parameters:

`INPUT phMessage AS HANDLE`

`OUTPUT phInMessageTarget AS HANDLE`

Returns: Nothing (procedure)

Notes: None

.offEnd (browser.p)

Triggers code for OFF-END trigger of SmartDataBrowse.

Parameters: None

Returns: Nothing (procedure)

Notes: None

offHome (browser.p)

Triggers code for OFF-HOME trigger of SmartDataBrowse.

Parameters: None

Returns: Nothing (procedure)

Notes: None

okObject (containr.p)

Save and close an object (Ok action)

Parameters: None.

Returns: Nothing (procedure).

Notes:

- If this is the window container or a virtual container then override and do **not** call SUPER. If not, then save and commit all Container-Targets and destroy if no errors occurred. Published from containerTargets or called directly.
- There is a slight overhead in this construct as destroyObject (called from exitObject -> apply “close”) does a publish “confirmExit”, which really is unnecessary after this has published “confirmOk”. The reason is that destroyObject might be called directly.
- We currently have to call exitObject as the appbuilder wait-for protests if we destroy directly. Even apply ‘close’ to target-procedure does not trigger the wait-for. It seems as this has to be fired from the actual instance. (exitObject should be local in all container instances) This might very well be a problem for application wait-for as well.

okObject (datavis.p)

Passes an ok request to its container

Parameters: None.

Returns: Nothing (procedure).

Notes: The convention is that the standard routine always passes an cancel, ok, or exit request to its CONTAINER-SOURCE. The container that is actually able to initiate the OK should override this and NOT call SUPER.

okToContinueProcedure (datavis.p)

Check whether the visual object has modified screen data or its data-source has uncommitted changes. If either of these conditions is true we issue Yes-No-Cancel questions allowing the user to choose how to proceed. The choices are: save/commit and continue; cancel/undo and continue; not continue.

Parameters:

`INPUT pcAction AS CHARACTER`

Used to retrieve the correct message. Valid values are

‘Exit’ -- Exit/close application window

‘ ’ [the empty string] -- Continue

‘Commit’ -- Commit (yes, no, cancel)

‘Undo’ -- Undo (no save option)

‘OK’ -- No question: save and commit

‘Cancel’ -- No question: cancel and undo

`OUTPUT plAnswer AS LOGICAL`

The user’s response to the question, if any

Returns: Nothing (procedure).

Notes: confirmExit and confirmOpenQuery call this function. This is copied from okToContinue function in datavis.p for use with Dynamics.

onChoose (panel.p)

Read action and parameter from Action class (Repository) and publishes the call built from them.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action to be undertaken

Returns: Nothing (procedure).

Notes: None.

onChoose (toolbar.p)

Persistent trigger code for dynamic menu and toolbar objects.

Parameters:

```
INPUT pcName AS CHARACTER
```

The Action identifier.

Returns: Nothing (procedure)

Notes: None

oneObjectLinks (smart.p)

Adds linkage for some object.

Parameters:

```
INPUT hObject AS HANDLE
```

Handle of the object to be linked.

Returns: Nothing (PRIVATE procedure.)

Notes: Called by displayLinks.

onMenuDrop (toolbar.p)

Logic to execute when a submenu is dropped.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action's unique identifier.

Returns: Nothing (procedure)

Notes: Added as a persistent trigger when the submenu is created.

onValueChanged (toolbar.p)

Persistent trigger for toggle menu-items.

Parameters:

```
INPUT pcAction AS CHARACTER
```

Name of action.

Returns: Nothing (procedure)

Notes: Added as a persistent trigger when the submenu is created. Currently supports only logical properties.

openDataQuery (data.p)

Opens the SmartDataObject's database query based on the current WHERE clause.

Parameters: None

```
INPUT pcPosition AS CHARACTER
```

"First" or "Last".

Returns: LOGICAL

Notes: Currently used by SBOs after new temp-tables have been fetched.

openQuery (data.p)

Opens the DataQuery.

Parameters: None

Returns: LOGICAL

Notes: None

openQuery (query.p)

General logic for opening the database query either for a DataObject or for another query object.

Parameters: None

Returns: LOGICAL

Notes: The QueryString property that is modified by the addQueryWhere, assignQuerySelection and setQuerySort methods will be used (if it is not blank) in the QUERY–PREPARE method before opening the query.

openQuery (sbo.p)

As a wrapper for fetchContainedData(), allows you to use the familiar SmartDataObject calling sequence.

Parameters: None

Returns: LOGICAL

Notes: None

ownerElement (xml.p)

Returns the OwnerElement corresponding to `pdAttributeNode`.

Parameters:

```
INPUT pdAttributeNode AS DECIMAL
```

Returns: DECIMAL

Notes: None

pageNTargets (containr.p)

Returns a comma-separated list of the objects on the specified page for this container.

Parameters:

```
INPUT phTarget AS HANDLE

INPUT piPageNum AS INTEGER
```

Returns: CHARACTER

Notes:

- The actual property pageNTarget keeps track of SmartObject page assignments within a Container. It is stored as a comma-separated list of entries, where each entry consists of an object handle in string format and its page number, separated by a vertical bar. The Target-Procedure is passed as a parameter because this function is only invoked locally, not in the Target-procedure.
- This function is only used internally by the ADM paging code.

parentNode (xml.p)

Returns the ID of *pdNode*'s parent.

Parameters:

```
INPUT pdNode AS DECIMAL
```

ID of the node whose parent is wanted.

Returns: DECIMAL node ID of the parent.

Notes: None

passThrough (containr.p)

Acts as an intermediary for dynamic links that need the pass-through mechanism, either to get an event from an object outside a container to one inside it or vice versa.

Parameters:

`INPUT pcLinkName AS CHARACTER`

The link (event) name to be passed on.

`INPUT pcArgument AS CHARACTER`

A single character-string argument.

Returns: Nothing (procedure)

Notes: To use this for single pass-through events, define a PassThrough link from the Source to the intermediate container, and define the actual dynamic link from the container to the Target. Then PUBLISH "PassThrough" (*`LinkName, Argument`*) to send the event.

populateTree (treeview.p)

Controls the population of the TreeView. Creates the node given in the argument and all its descendents.

Parameters:

`INPUT phTable AS HANDLE`

The handle of the temp-table containing the data (defined in tvcontroller.i)

`INPUT pcStartNodeKey AS CHARACTER`

The highest-level node

Returns: Nothing (procedure).

Notes:

- This routine is called recursively to descend the tree pointed to by phTable.
- The temp-table tTreeData includes the following fields: (as defined in tvcontroller.i)
  - node_key — Unique key of node
  - parent_node_key — Key of either parent node or sibling node (depending on node_insert)
  - node_label — Label of node that appears in treeview
  - private_data — Information stored for the node in the node's TAG property
  - image — Relative path and filename of the image of the node
  - selected_image — Relative path and filename of the node when it is selected. If blank, no special image is used.
  - node_insert — Specifies where to insert the node, relative to the parent node:

    0 - As first node at same level as the parent_node_key

    1 - As last node at same level as the parent_node_key

    2 - After (next) the parent_node_key

    3 - Before (previous) the parent_node_key

    4 - As child of parent_node_key
  - node_sort — TRUE if the node is to be shown sorted within its sibling group. All of this node's siblings must have this option set for sorting to be complete. Default is FALSE.
  - node_expanded — TRUE if the node is to be expanded upon adding it to treeview. Default is FALSE.
  - node_checked — TRUE if the node is to be shown checked. Default is FALSE. (The property 'ShowCheckBoxes' must be set TRUE to enable this option).

postCreateObjects (sbo.p)

Runs at the end of createObjects(), after all contained Objects have been created but before they have been initialized. Sets various properties that are dependent on knowing the handles and Instance Properties of all contained objects, and fetches property settings from the server-side SmartBusinessObject.

Parameters: None

Returns: Nothing (procedure)

Notes: None

postTransactionValidate (datalogic.i)

The optional validation procedure called automatically by serverCommit after all updates complete and the transaction block ends.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

prepareErrorsForReturn (data.i)

Appends the RETURN-VALUE to the list of logged errors, formats the string for return to the client. Called from serverCommit.

Parameters:

```
INPUT pcReturnValue AS CHARACTER

INPUT pcASDivision AS CHARACTER

INPUT-OUTPUT pcMessages AS CHARACTER
```

Returns: Nothing (procedure)

Notes: PRIVATE procedure.

prepareErrorsForReturn (sbo.p)

Appends the RETURN-VALUE from the user-defined transaction validation procedure or other update-related error to the list of any errors already in the log. Formats this string to prepare for returning it to the client.

Parameters:

```
INPUT pcReturnValue AS CHARACTER

INPUT pcASDivision AS CHARACTER

INPUT-OUTPUT pcMessages AS CHARACTER
```

Returns: Nothing (procedure)

Notes: Invoked internally from serverCommitTransaction().

prepareQueriesForFetch (sbo.p)

Prepares queries in SmartDataObjects for a fetch of data from the server.

Parameters:

```
INPUT pcObjectName AS CHARACTER

INPUT pcOptions AS CHARACTER

OUTPUT pocQueries AS CHARACTER

OUTPUT poctempTables AS CHARACTER
```

Returns: Nothing (procedure)

Notes:

- This procedure exists in order to have common logic for fetchContainedData and fetchContainedRows.
- Cannot be used across sessions because temp-table handles are concatenated in a list. Tests indicate that it is faster to add all the handles to a list than have a procedure with 20 output parameters for the temp-table handles.

prepareQuery (data.p)

Prepares the SmartDataObject's database query based on the passed parameter. This procedure isolates the query preparation for client/server, keeping it away from the logic needed to clean and track the QueryString property.

Parameters:

```
INPUT pcQuery AS CHARACTER
```

A complete query expression.

Returns: LOGICAL

Notes: This function simplifies certain behavior in previous releases.

prepareQuery (query.p)

Internal function. Prepares the database query for some querying object such as a SmartDataObject.

Parameters:

```
INPUT pcQuery AS CHARACTER
```

The complete query expression

Returns: LOGICAL

Notes:

The main purpose for this is to prepare a query on an AppServer. You can do this in two different ways:

- addQueryWhere, assignQueryWhere manipulates a client-side property that openQuery will check and use as input to this method.
- setQueryWhere will call this method and blank properties used by 1 in order to make openQuery NOT call this.

preTransactionValidate (datalogic.i)

The optional validation procedure called automatically by serverCommit and before entering the transaction block.

Parameters: None.

Returns: Nothing (procedure).

Notes: Use this in place of TransactionValidate (maintained for backward compatibility).

printToCrystal (data.p)

Transfers the data from the SDO to Crystal.

Parameters:

```
INPUT pcFieldList AS CHARACTER
```

The list of fields to be transferred, or the empty string to mean ‘all’.

```
INPUT plIncludeObj AS LOGICAL
```

TRUE if object fields are to be included

```
INPUT piMaxRecords AS INTEGER
```

The maximum number of records to transfer

Returns: Nothing (procedure).

Notes: Always excludes rowobject specific fields, e.g. RowNum, RowIdent, RowMod. Uses tableout procedure.

processAction (visual.p)

Generic procedure to process keyboard and mouse actions for visual objects.

Parameters:

```
INPUT pcAction AS CHARACTER
```

The action to be processed.

Returns: Nothing (procedure).

Notes: Typically a generic event will be put in visualcustom.i that will run this procedure in order to do the work. It can not do the work normally itself as container handles, etc. are not valid yet. Also this keeps the code size down.

processCDataSection (xml.p)

Processes a CDATA-Section node.

Parameters:

```
INPUT phText AS HANDLE
```

```
INPUT pcPath AS CHARACTER
```

Returns: Nothing (procedure)

Notes: This method just passes an event with path and value information.

processComment (xml.p)

Processes a comment node.

Parameters:

```
INPUT phText AS HANDLE
```

```
INPUT pcPath AS CHARACTER
```

Returns: Nothing (procedure)

Notes: This method does nothing and must be overridden if you need to process comments.

processDestinations (consumer.p)

Creates tDestination temp-table records from the Destinations, Subscriptions, and Selectors property values.

Parameters: None

Returns: Nothing (procedure)

Notes: createConsumers() uses tDestination temp-table records when creating the consumers that will monitor the individual destinations. You can override this with a version of processDestinations() that creates its own tDestination temp-table records rather than using the property values.

processDocument (xml.p)

Starts the processing of the current document.

Parameters: None

Returns: Nothing (procedure)

Notes: None

processElement (xml.p)

Processes an element document.

Parameters:

`INPUT phNode AS HANDLE`

Current x-noderef handle.

`INPUT pcPath AS CHARACTER`

Current Documentpath (This is the logical path and not the numbered XPath.)

Returns: Nothing (procedure)

Notes:

- Calls startEvent before the children are processed and endEvent after.
- Attributes are NOT processed in xml.p, but all the attribute's names are passed as parameters to the startEvent. The developer or super-procedure extension is responsible for processing data from attributes.

processFileRefs (router.p)

Creates tFileReference temp-table records from ExternalRefList and InternalRefList property values.

Parameters: None

Returns: Nothing (procedure)

Notes: None

processMappings (b2b.p)

Creates tMapping temp-table records from DirectionList, NameList, SchemaList, DocTypeList, DestinationList, ReplyReqList, and ReplySelectorList property values.

Parameters: None

Returns: Nothing (procedure)

Notes: None

processMessages (b2b.p)

Called from receiveHandler() when an error occurs.

Parameters:

```
INPUT pcMessages AS CHARACTER
```

Chr(3) and chr(4) delimited list, as returned from fetchMessages().

Returns: Nothing (procedure)

Notes: Passes the data from fetchMessages() so the user does not have to do this. Otherwise, the default is exactly as in ShowDataMessages() in other ADM objects. This is called from ReceiveHandler() so RETURN ERROR or "ADM-ERROR" will make sure that the message is not acknowledged.

processRoot (xml.p)

Starts processing the elements of the document.

Parameters: None

Returns: Nothing (procedure)

Notes: This is just a starting point without events.

processText (xml.p)

Processes a text node.

Parameters:

```
INPUT phText AS HANDLE
```

```
INPUT pcPath AS CHAR
```

Returns: Nothing (procedure)

Notes: Passes the event with path and value information.

produceAttributes (b2b.p)

Produces attributes of schema nodes.

Parameters:

`INPUT piParentSchemaNode AS INTEGER`

`INPUT pdOwnerNode AS DECIMAL`

`INPUT phDataSource AS HANDLE`

Returns: Nothing (procedure)

Notes: None

produceChildren (b2b.p)

Produces all child nodes of an element/document.

Parameters:

`INPUT piParentSchemaNode AS INTEGER`

Schema parent node.

`INPUT pdParentNode AS DECIMAL`

Actual parent node.

`INPUT phDataSource AS HANDLE`

LINKed data source of parent.

Returns: Nothing (procedure)

Notes: None

produceDocument (b2b.p)

Creates a document and runs produceChildren() on it.

Parameters: None

Returns: Nothing (procedure)

Notes: None

propertyType (smart.p)

Locates the "set" property function for the specified property name either locally or in a SUPER procedure, and returns its data type.

Parameters:

```
INPUT pcPropName AS CHARACTER
```

Property name.

Returns: CHARACTER (The data type of the property.)

Notes: Generally used internally by the ADM.

Examples:

```
/* This code, adapted from the addLink procedure, determines whether a
   link is permitted to have multiple Targets based on whether the
   supporting property function accepts a handle (for a single object
   handle) or a character string (for a comma-separated list of
   handles. */

IF dynamic-function('propertyType':U, INPUT pcLinkType + "Source":U) =
"CHARACTER":U THEN
  RUN modifyListProperty ('ADD':U º )   /* To add the string to a list. */
ELSE dynamic-function('set':U + pcLinkType + "Source":U, INPUT phTarget).
```

pushTableAndValidate (data.i)

When running from a SmartBusinessObject, acts as a wrapper for preTransactionValidate and postTransactionValidate procedures.

Parameters:

```
INPUT pcValType AS CHARACTER
```

Type of procedure needing the wrapper. Valid values are "Pre" and "Post."

```
INPUT-OUTPUT TABLE FOR RowObjUpd
```

Returns: RETURN-VALUE

Notes: None

putParam (tvcontroller.p)

Parameters:

```
INPUT pcParams AS CHARACTER
```

Must be a valid parameter-list format

```
INPUT pcParamName AS CHARACTER


INPUT pcValue AS CHARACTER
```

Returns: CHARACTER

Notes: None.

queryOpened (select.p)

Subscribes to dataSource parameters.

Parameters: None

Returns: Nothing (procedure)

Notes: SmartSelect needs to know that the query has changed, since this cannot be detected through an ordinary publish with "dataAvailable."

queryPosition (datavis.p)

Captures “state” events for the associated query in the object’s data source.

Parameters:

```
INPUT pcState AS CHARACTER
```

Possible values are: FirstRecord (first record); NotFirstOrLast (not the first or last record); LastRecord (last record); NoRecordAvailable (no record); OnlyRecord (only one record)

Returns: Nothing (procedure)

Notes:

- When any state begins "NoRecordAvail" it means to disable; all others (FirstRecord, NotFirstOrLast, LastRecord, OnlyRecord) mean to enable if in "Save" mode (SaveSource property is set to yes).
- Sets the RecordState property to "NoRecordAvailable" when the state is equivalent to that; but for any other state, sets the value to "RecordAvilable."

queryPosition (panel.p)

Captures “state” events for the associated Query in the Panel’s NavigationTarget. Invokes the setPanelState function, which stores the new state in the object’s PanelState property and then invokes the setButtons procedure to change the Panel.

Parameters:

```
INPUT pcState AS CHARACTER
```

Panel State

Returns: Nothing (procedure)

Notes: For an Update panel, first/last/notfirstorlast/only are all forms of recordAvailable. This event will be ignored if the source of the event is hidden.

queryPosition (sbo.p)

Receives the queryPosition event from a contained SmartDataObject and passes it on to the appropriate Navigation-Source or other object.

Parameters:

```
INPUT pcPosition AS CHARACTER
```

Returns: Nothing (procedure)

Notes: None

queryPosition (toolbar.p)

Captures state events for the associated query in the Panel's NAVIGATION-TARGET. Invokes the setPanelState() function to store the new state in the object's PanelState property. Then invokes the setButtons procedure to change the Panel.

Parameters:

```
INPUT pcState AS CHARACTER
```

State of the Panel.

Returns: Nothing (procedure)

Notes: For an UPDATE panel, first/last/notfirstorlast/only are all forms of recordAvailable.

receiveHandler (b2b.p)

Handles receiving a message.

Parameters:

```
INPUT phMessage AS HANDLE
```

Handle of the JMS message being received.

Returns: Nothing (procedure)

Notes: The `xml.p` SUPER creates an XML document and loads it from the message.

receiveHandler (xml.p)

Handles receiving a message

Parameters:

```
INPUT phMessage AS HANDLE
```

Returns: Nothing (procedure)

Notes: phMessage is the handle of the JMS message being received. An XML document is created and it is loaded with the memptr from the message. The Document is NOT processed by default. The developer can override this to start processing the document either by calling processDocument or getElementsByTagName or other equivalent methods.

receiveReplyHandler (xml.p)

Handles any reply to a message that has been sent.

Parameters:

```
INPUT phMessage AS HANDLE
```

Handle of the JMS reply message being received.

Returns: Nothing (procedure)

Notes: An override of this procedure should get the Correlation ID of the message to synchronize it to the original message being replied to. It should also get any needed property values and the body of the reply message.

refetchDBRow (query.p)

Re-reads the current database records to get any field values that might have been changed by update triggers, and repopulates the RowObjUpd row with those values.

Parameters:

```
INPUT phRowObjUpd AS HANDLE
```

The handle of the Update temp-table buffer to be used. Note that this is required because the caller uses two different buffers in some cases.

Returns: Nothing (procedure)

Notes: None

refreshBrowse (browser.p)

Published from dataSource when APPEND display is not enough or not required.

Parameters: None

Returns: Nothing (procedure)

Notes: None

refreshBrowse (sbo.p)

Receives the refreshBrowse event from a contained SDO and passes it on to the appropriate DataTarget.

Parameters:

```
INPUT pcPosition AS CHARACTER
```

Returns: Nothing (procedure).

Notes: None.

refreshChildDependancies (combo.p)

Refreshes any children of the Combo identified in the argument.

Parameters:

```
INPUT pcFieldName AS CHARACTER
```

The possible parent

Returns: Nothing (procedure).

Notes: None.

refreshCombo (combo.p)

Runs displayCombo in TARGET-PROCEDURE. This is usually triggered from a value-changed event of a parent combo.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

registerLinkedObjects (sbo.p)

Register objects in the ObjectMapping and other properties. This currently applies to navigationSources and Datatargets. Currently we assume that UpdateSources are DataTargets also.

Parameters: None.

Returns: Nothing (procedure).

Notes: This procedure is used to register objects that have already been initialized when the SBO is initialized. The SBO also subscribes to ‘registerObject’ in these objects, which they publish at initialization.

refreshObject (select.p)

Refreshes the data in a SmartSelect.

Parameters: None

Returns: Nothing (procedure)

Notes: openQuery in the datasource publishes “queryOpened” which is Subscribed to run this (in initializeSelection). Also defined as PERSISTENT trigger on F5 of the selection widget.

refreshRow (data.p)

Retrieves the current database values for a row already in the RowObject table.

Parameters: None

Returns: Nothing (procedure)

Notes: PUBLISHes dataAvailable ("SAME") to cause a SmartDataViewer or Browser to display the latest values. Since the database records are never locked on read, this procedure can be used to fetch the latest values upon a request from an application, but cannot guarantee that they will not change before an update is done.

refreshRowident (query.p)

Returns a refreshed list of rowids for the read-only tables in the current query.

Parameters:

```
INPUT pcRowident AS CHARACTER
```

RowIdent of a row in the current query.

Returns: CHARACTER

Notes:

- Used internally by refetchDbRow in order to make changes to foreignfields reflected in correctly joined tables. The refresh applies to read-only tables that are at the end of the join (no updatable tables further down in the join). The first Rowid is **not** refreshed. Be sure to use SDOs correctly.

- Only one copy of the record that is being updated in the query, which means one-to-one updates if many updatable records. There are several changes that still require an open query to be reflected.
- Changes in a child record that changes or makes a parent relationship invalid is not reflected and requires a complete open query.
- Changes that make an inner-join invalid will blank out Rowids for **all** read-only tables, even if some of them were still valid. Note that if the query was reopened, this row would disappear from the query. Normally an application would check that no invalid keys are entered, but a value could also be changed so that it was filtered away.
- Two updatable tables will not reflect changes in the relationship to each other.

registerObject (sbo.p)

A general purpose "register" event published from objects at initialization. This object defines this as navigationSourceEvent and dataTargetEvent. The event is used to register objects in ObjectMapping and other properties. (Currently, assume that updateSources are also DataTargets.)

Parameters: None

Returns: Nothing (procedure)

Notes: None

releaseDBRow (query.p)

Releases the database records following an update.

Parameters: None

Returns: Nothing (procedure)

Notes: None

ReleaseDC (treeview.i)

Defined as EXTERNAL "user32".

Releases some device context.

Parameters:

`INPUT hWnd AS LONG`

The window handle

`INPUT hDC AS LONG`

The device context handle

`RETURN hDeviceConext AS LONG`

The device context handle

Returns: Nothing (external procedure)

Notes: None.

removeAllLinks (smart.p)

Removes all links for a SmartObject, normally as part of destroying a SmartObject procedure.

Parameters: None

Returns: Nothing (procedure)

Notes: Run automatically as part of destroyObject. Not normally expected to be run by user application code (see removeLink for a procedure to remove a single specific link).

removeForeignKey (query.p)

Removes the ForeignKey from the query string.

Parameters: None

Returns: LOGICAL

Notes: The ForeignKey consists of ForeignKeys and ForeignValues.

removeLink (smart.p)

Removes a specific link between two objects.

Parameters:

`INPUT phSource AS HANDLE`

Source procedure handle.

`INPUT pcLink AS CHARACTER`

Link type name.

`INPUT phTarget AS HANDLE`

Link target object handle.

Returns: Nothing (procedure)

Notes:

- This procedure both removes the handles of both Source and Target from the appropriate SmartObject properties (see addLink) and also does an UNSUBSCRIBE for each named event associated with the link.
- All SmartObject links are removed when an object is destroyed. This procedure could be used in application code to remove a specific link based on application requirements.

Examples:

```
/* This example removed the Update link between a visual SmartObject such
   as a SmartDataViewer and its associated SmartDataObject, perhaps in
   response to an application event or security check which should make

   the SmartDataObject non-updatable. */

hTarget = WIDGET-HANDLE(dynamic-function('linkHandles', INPUT

   'Update-Target':U)).
IF VALID-HANDLE(hTarget) THEN
   RUN removeLink(THIS-PROCEDURE, 'Update':U, hTarget).
```

removePageNTarget (containr.p)

Removes an object from the list of Targets on a page.

Parameters:

`INPUT phTarget AS HANDLE`

The handle of the object to be removed.

`INPUT piPage AS INTEGER`

The page number of the object to be removed.

Returns: Nothing (procedure)

Notes: Run from removeAllLinks for objects not on Page 0. Not intended to be run from application code.

removeQuerySelection (query.p)

Removes query selection criteria from the QueryString property that were added by assignQuerySelection.

Parameters:

`INPUT pcColumns AS CHARACTER`

A comma-separated list of column names that are the subject part of a phrase to be removed.

`INPUT pcOperators AS CHARACTER`

The operator (one for all columns):

- A blank defaults to (EQ).
- A slash is used to define an alternative string operator (EQ/BEGINS, and so forth).
- A comma-separated list for each column/value.

Returns: LOGICAL

Notes: This procedure modifies the QueryString property and is designed to run on the client and called several times before QueryString is used in a QUERY–PREPARE method to modify the database query. openQuery will prepare the query using the QueryString property. The removal of the expression is done by finding the value of which position and length is stored in the QueryColumns property.

removeQuerySelection (sbo.p)

Separates the Columns by SmartDataObject and passes columns and operators on to the appropriate SmartDataObjects. SmartBusinessObject version of this where-clause function.

Parameters:

`INPUT pcColumns AS CHARACTER`

Comma-separated list of column names, qualified by their ObjectNames, that are the subject part of a phrase to be removed.

`INPUT pcValues AS CHARACTER`

Comma-separated list of the corresponding values.

`INPUT pcOperators AS CHARACTER`

The operator, which might be:

- A blank defaults to (EQ).
- A slash (/) is used to define an alternative string operator (EQ/BEGINS, etc.).
- A comma-separated list corresponding to columns and values lists.

Returns: LOGICAL

Notes: All columns must be qualified by their SmartDataObject ObjectName as TableName; this will be replaced with RowObject when the columns are passed on to the SmartDataObject.

removeSpace (filter.p)

Backspaces, deletes, and sets the field to unmodified. Keeps it out of the selection criteria.

Parameters: None

Returns: Nothing (procedure)

Notes: Start persistent in initializeObject. Note that `filter.i` has the following trigger anywhere on frame: on value-changed > u10 > unblankFillin trigger resets format when all fields have been blanked (emptied).

replyHandler (producer.p)

Handles replies to messages.

Parameters:

```
INPUT phReply AS HANDLE

INPUT phConsumer AS HANDLE

OUTPUT phResponse AS HANDLE
```

Returns: Nothing (procedure)

Notes: None

repositionDataSource (select.p)

Repositions the current dataSource according to the key field. Conditionally called from setDataValue() and valueChanged().

Parameters:

```
INPUT pcValue AS CHARACTER
```

Returns: LOGICAL

Notes: None

repositionObject (smart.p)

Adjusts the position of container objects.

Parameters:

```
INPUT pdRow AS DECIMAL
```

```
INPUT pdCol AS DECIMAL
```

Returns: Nothing (procedure)

Notes: None

repositionObject (toolbar.p)

Overrides default reposition. Because the coordinates are NOT assigned in design, the position only changes through direct manipulation and not when dropped in the container.

Parameters:

```
INPUT pdRow AS DECIMAL
```

```
INPUT pdCol AS DECIMAL
```

Returns: Nothing (procedure)

Notes: The file `toolbar.i` defines EXCLUDE-repositionObject (for `smart.i`).

repositionObject (treeview.p)

Super override for Treeview object repositioning.

Parameters:

```
INPUT pdRow AS DECIMAL
```

New row

```
INPUT pdCol AS DECIMAL
```

New column

Returns: Nothing (procedure).

Notes: None.

repositionRowObject (data.p)

Positions the current-record pointer to the record passed as argument. Used during updates to prevent errors caused by the user moving the current-record pointer before completing the update.

Parameters:

```
INPUT pcRowIdent AS CHARACTER
```

The identifier of the target record

Returns: LOGICAL

Notes: We need to use entry 1 of the rowident which is the rowid of the rowobject temp-table record. This is NOT using reposition because we do not want the browser to reposition. This can fail only if the query has been reopened and the rowobject temp-table rebuilt, meanwhile, changing the rowids. It is to be hoped that this would not happen.

repositionRowObject (sbo.p)

Passes the rowident on to the contained DataObject that manages that data. SBO version of the routine.

Parameters:

```
INPUT pcRowIdent AS CHARACTER
```

Semicolon-separated list, one entry for each ContainedDataObject. The visual object's getRowIdent returns this correctly.

Returns: LOGICAL

Notes: None.

resetCommit (panel.p)

Resets the commit target actions.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

resetFields (filter.p)

Resets all the fields to the previously applied filter values.

Parameters: None

Returns: Nothing (procedure)

Notes: The field values and operator values are retrieved form the actual filter-targets query.

resetNavigation (panel.p)

Resets the navigation target actions.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

resetQuery (sbo.p)

Resets the query for the identified SmartDataObject. If the query being reset belongs to the Master object, the operation brings it to its default state (Open Query). If the query belongs to some other object, the reset operation clears the QueryString property.

If pcObject is not specified, this routine resets the queries for all contained SmartDataObjects.

Parameters:

```
INPUT pcObject AS CHARACTER
```

SmartDataObject ObjectName.

Returns: LOGICAL

Notes: This function is here because setQueryWhere is not supported for SmartBusinessObjects, and running setQueryWhere('') would be the way to do it otherwise.

resetRecord (browser.p)

SmartDataBrowser version of resetRecord. Redisplays the original field values retrieved from the SmartDataObject.

Parameters: None

Returns: Nothing (procedure)

Notes:

- Invokes resetRecord in `datavis.p` (RUN SUPER) for a SmartDataBrowser connected to a SmartDataObject as its Data source. If the SmartDataBrowser has its own database query, invokes displayFields.
- Invoked when the reset button is chosen in an Update SmartPanel or in a SmartToolbar.
- Customized when additional processing is needed during redisplay of a record, perhaps to reset other related SmartObjects or standard object fields.

resetRecord (datavis.p)

A general code for reset operation. Redisplays the original field values retrieved from the SmartDataObject by invoking the colValues() function for the fields that are displayed.

Parameters: None

Returns: Nothing (procedure)

Notes:

- Invoked when the Reset button is pressed on an Update SmartPanel or a SmartToolbar.
- Publishes "resetRecord" in the case of a GroupAssign. Sets DataModified property to "?".

resetTableio (panel.p)

Encapsulate ALL tableio settings

Parameters: None.

Returns: Nothing (procedure).

Notes: Can probably replace ALL other events except

- linkChanged('inactive').
- updateState('updateComplete') and Tableiomode = 'update' 'backfire' of updateMode()

resetTargetActions (panel.p)

Resets action sensitivity and visibility.

Parameters:

```
INPUT pcLink AS CHARACTER
```

The link type whose actions are to be reset

Returns: Nothing (procedure).

Notes: Overridden in toolbar.p with use of temp-tables and additional logic for alternate image.

resetTargetActions (toolbar.p)

Resets target actions of some kind for an object.

Parameters:

```
INPUT pcLink AS CHARACTER
```

The kind of actions to reset

Returns: Nothing (procedure).

Notes: None.

resetToolbar (toolbar.p)

Resets toolbar actions by calling resetTargetActions.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

resizeBrowse (browser.p)

Changes the size of the BROWSER WIDGET-HANDLE IN APPEND SmartDataBrowser.

Parameters:

`INPUT pd_height AS DECIMAL`

The desired height (in rows).

`INPUT pd_width AS DECIMAL`

The desired width (in columns).

Returns: Nothing (procedure)

Notes: None

resizeObject (browser.p)

Changes the size of the SmartDataBrowser.

Parameters:

`INPUT pdHeight AS DECIMAL`

A decimal value that represents the new height of the SmartDataBrowser.

`INPUT pdWidth AS DECIMAL`

A decimal value that represents the new width of the SmartDataBrowser.

Returns: Nothing (procedure)

Notes:

- Invoked at design time when the SmartDataBrowser is resized. Can be invoked at run time to change the size of the SmartDataBrowser.
- The presence of the resizeObject procedure causes the AppBuilder to generate a statement in the adm–create–objects procedure for a SmartContainer setting the size of a contained SmartDataBrowser. The presence of resizeObject also signals the AppBuilder to allow the SmartDataBrowser to be resized at design time.

resizeObject (combo.p)

Resizes the Combo by deleting the widget, resizing the frame, and recreating the widget in the new size.

Parameters:

`INPUT pidHeight AS DECIMAL`

New height of the widget

`INPUT pidWidth AS DECIMAL`

New width of the widget

Returns: Nothing (procedure).

Notes: None.

resizeObject (field.p)

Resizes the field by looping through all fields in the Frame and resizing any that do not have ‘NO-RESIZE’ in the PRIVATE-DATA property.

Parameters:

`INPUT pidHeight AS DECIMAL`

New height

`INPUT pidWidth AS DECIMAL`

New width

Returns: Nothing (procedure).

Notes: No resizing will be done by default. For this you need to create an override procedure in your SmartDataField with your own code, specific to your SmartDataField.

resizeObject (lookup.p)

Resize the Lookup by deleting the widget, resizing the Frame, and recreating the widget.

Parameters:

`INPUT pidHeight AS DECIMAL`

The new height

`INPUT pidWidth AS DECIMAL`

The new width

Returns: Nothing (procedure).

Notes:

resizeObject (panel.p)

Changes the size and shape of the panel. This routine spaces the buttons to fill the available space.

Parameters:

`INPUT pd_height AS DECIMAL`

The desired height (in rows).

`INPUT pd_width AS DECIMAL`

The desired width (in columns).

Returns: Nothing (procedure)

Notes: Used internally. Calls to resizeObject are generated by the AppBuilder in adm–create–objects for objects that implement it. Having a resizeObject procedure is also the signal to the AppBuilder to allow the object to be resized at design time.

resizeObject (select.p)

Resizes the SmartSelect.

Parameters:

`INPUT pidHeight AS DECIMAL`

New height of component

`INPUT pidWidth AS DECIMAL`

New width of component

Returns: Nothing (procedure)

Notes: The procedure deletes the current widget, resizes the frame, and re-creates the widget.

resizeObject (toolbar.p)

Overrides the size after a resize. The only allowed sizing is shrinking to the last button.

Parameters:

```
INPUT pd_height AS DECIMAL
```

```
INPUT pd_width AS DECIMAL
```

Returns: Nothing (procedure)

Notes: Used internally. Calls to resizeObject are generated by the AppBuilder in adm-create-objects for objects that implement it. Having a resizeObject procedure is also the signal to the AppBuilder to allow the object to be resized at design time.

resizeObject (treeview.p)

Resizes a SmartTreeView object.

Parameters:

```
INPUT pdHeight AS DECIMAL
```

New height

```
INPUT pdWidth AS DECIMAL
```

New width

Returns: Nothing (procedure).

Notes: None.

resizeWindow (containr.p)

Respond to a resize event from user event or container targets.

Parameters: None.

Returns: Nothing (procedure).

Notes: The current functionality just resizes the frame according to the window size. This was added to make the call from the toolbar's resizeObject after it expands a window have some default functionality, but this is mostly a placeholder for logic to resize all contained objects, currently implemented in ry/uib/rydyncontw.w.

resolveBuffer (query.p)

Resolves the correct qualified buffer name of the passed buffer reference.

Parameters:

```
INPUT pcBuffer AS CHARACTER
```

Buffer name, qualified or unqualified

Returns: CHARACTER

Notes:

- Returns blank if the buffer cannot be resolved in the SDO. Returns unknown if the table reference is ambiguous (More than one table in the SDO matches the unqualified input parameter).
- Used internally (columnTable and others) to resolve cases where the passed column name's qualification is different from the object's.
- There is no reference to the query handle in order to resolve this on the client.

restartServerObject (appserver.p)

When a SmartDataObject is split and running stateless on an AppServer, it is shut down after each use and then restarted for the next. restartServerObject is run on the client to restart the SmartDataObject on the server.

Parameters: None

Returns: Nothing (procedure)

Notes:

- The startServerObject runs on the first connection.
- Both start- and restart- runs runServerObject, which does the actual run and the optional initialize.
- This procedure is considered to be an “internal event” and silently returns if the object is bound, or Appservice = ' ' and does not check if ASHasStarted is true (because this might not be damaging if there is no difference between start/restart). It is assumed to be called only when unbound and ASHasStarted is true, as when called from getAsHandle.
- getASHandle depends on the “silent error” when Appservice is blank for backwards compatibility with data and SBO classes (it actually checked ASDivision="client", but that is too restrictive). So, if errors are added here, make sure getAsHandle deals with (removes) them before returning a "?".

restartServerObject (data.p)

Restarts the SmartDataObject on the server.

Parameters: None

Returns: Nothing (procedure)

Notes: When a SmartDataObject is split and running Stateless on an AppServer, it is shut down after each use and then restarted for the next. restartServerObject is run on the client to restart the SmartDataObject on the server.

restartServerObject (sbo.p)

When a SmartBusinessObject is split and running stateless on an AppServer, it is shut down after each use and then restarted for the next. restartServerObject is run on the client to restart the SmartBusinessObject on the server.

Parameters: None

Returns: Nothing (procedure)

Notes: This override is for error handling to show error message and return "adm-error".

retrieveFilter (data.p)

Obtains and restores the filter to its defined state. This depends on links that are only available on the client side.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

returnFocus (smart.p)

Returns focus to the containing window.

Parameters:

```
INPUT hTarget AS HANDLE
```

A handle to the target procedure object.

Returns: Nothing (procedure)

Notes: None

returnParentFieldValues (combo.p)

Loops through all field handles.

Parameters:

```
OUTPUT pcNewQuery AS CHARACTER
```

Returns: Nothing (procedure).

Notes: None.

returnParentFieldValues (lookup.p)

Parameters:

```
OUTPUT pcNewQuery AS CHARACTER
```

Returns: Nothing (procedure).

Notes: None.

reviewMessages (smart.p)

Returns a delimited list of all messages without removing them from the log.

Parameters: None

Returns: CHARACTER (Specially delimited message list.)

Notes:

- reviewMessages is not normally expected to be used by application code. The showDataMessages function can be used (and customized if desired) to parse this specially formatted message string into a series of error messages.

- The message list is delimited by CHR(3); within each message, the Message Text, the Field (if any), and the Table (if any) are delimited by CHR(4).
- reviewMessages is intended to be used in situations where it is necessary to examine messages in the error log without deleting them, so that some other procedure can later process them. Use the similar function fetchMessages to read messages and simultaneously delete them from the log. Note that if all that is needed is to determine if there are any messages in the log, the anyMessage function can be used.

routeBytesMessage (router.p)

Takes a procedure that has a BytesMessage, loads the message, and sends the document to a SmartB2BObject.

Parameters:

`INPUT phMessage AS HANDLE`

A procedure handle to the object with the BytesMessage and a corresponding getMemptr function to retrieve the message.

`OUTPUT phInMessageTarget AS HANDLE`

The procedure handle of the SmartB2BObject.

Returns: Nothing (procedure)

Notes: Used for backward compatibility, but it is possible to override if message type requires a different routing. See obtainInMsgTarget (`router.p`) for details.

routeDocument (router.p)

Routes the documentHandle to a SmartB2BObject, including starting the SmartContainer with the SmartB2B if required, and giving it the loaded schema.

Parameters:

`OUTPUT pohInMessageTarget AS HANDLE`

The SmartB2B that received the document.

Returns: Nothing (procedure)

Notes: None

routeMessage (router.p)

Takes a procedure that has a message, loads the message, and sends the document to a B2B object and returns its handle.

Parameters:

`INPUT phMessage AS HANDLE`

A procedure handle to the object with the JMS message or any object with getMessageType and a corresponding get function to retrieve the message.

`OUTPUT pohInMessageTarget AS HANDLE`

The procedure handle of the SmartB2B that received the document.

Returns: Nothing (procedure)

Notes: Called directly from obtainInMesages or from specific route *`MessageType`* procedures.

rowAvailable (data.p)

Checks RowObject availability. Encapsulates the different query position alternatives required to check for availability.

Parameters:

`INPUT pcDirection AS CHARACTER`

- NEXT: Is there a next record available.
- PREV: Is there a previous record available.
- ' ', ?, or CURRENT: Current record.

Returns: LOGICAL

Notes: This can be used in loops to simplify logic when navigating.

rowDisplay (browser.p)

Event Handler for the ROW-DISPLAY event.

Parameters: None

Returns: Nothing (procedure)

Notes: None

rowidWhere (query.p)

Returns the ROWID (converted to a character string) of the first database query row satisfying the where clause. In the case of a join, only the rowid of the first table in the join will be returned and the expression in pcWhere can only reference that table.

Parameters:

```
INPUT pcWhere AS CHARACTER
```

The where clause to apply to the database query to fetch the first record whose ROWID will be returned.

Returns: CHARACTER

Notes: The ROWID is returned as a string both in anticipation of it being used as an argument to fetchRowIdent, and also to allow this function to be invoked from outside Progress.

rowidWhereCols (query.p)

Returns a list of ROWIDs and adds column/value pairs to the corresponding buffer's WHERE clause. Each buffer's expression is embedded in parentheses.

Parameters:

`INPUT pcColumns AS CHARACTER`

Column names (comma separated). Fieldname of a table in the query in the form of TBL.FLDNM or DB.TBL.FLDNM (only if qualified with db), (RowObject.FLDNM should be used for SDOs). If the fieldname is not qualified, it checks the tables in the TABLES property and assumes the first is a match.

`INPUT pcValues AS CHARACTER`

Corresponding Values (CHR(1) separated).

`INPUT pcOperators AS CHARACTER`

The operator (one for all columns):

- A blank defaults to (EQ).
- A slash is used to define an alternative string operator (EQ/BEGINS, and so forth).
- A comma-separated list for each column/value.

Returns: CHARACTER

Notes: None

rowNotFoundError (b2b.p)

Returns a "Record not found" error message with key fields.

Parameters:

`INPUT phDataSource AS HANDLE`

DataSource.

`INPUT pcKeyValues AS CHARACTER`

CHR(1) list with KeyValues corresponding to KeyFields.

Returns: CHARACTER

Notes: None

rowObjectState (data.p)

Passes on the flag, which indicates when a SmartDataObject row has been changed locally but not committed. This gets passed on until it reaches the Commit Panel or other first Commit–Source.

Parameters:

```
INPUT pcState AS CHARACTER
```

The new state.

Returns: Nothing (procedure)

Notes: This SmartDataObject version does not do anything itself with the state setting, but simply passes it on up the chain.

rowObjectState (toolbar.p)

Published from COMMIT-TARGET to tell the panel when to enable/disable itself. The panel enables itself when there are uncommitted changes, and disables itself at other times.

Parameters:

```
INPUT pcState AS CHARACTER
```

Valid values: "NoUpdates", "RowUpdated"

Returns: Nothing (procedure)

Notes: This could be a property, but at present the solution is to just check the state of the Commit button.

rowObjectValidate (datalogic.i)

The client-side validation procedure for RowObject.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

rowSelected (lookup.p)

Publish a lookupcomplete event in the container allowing developers to perform some action when a row is selected from the browser.

Parameters:

`INPUT pcAllFields AS CHARACTER`

Comma-separated list of found fieldnames

`INPUT pcValues AS CHARACTER`

Chr(1)-separated list of corresponding field values

`INPUT pcRowIdent AS CHARACTER`

Comma-separated list of ROWIDs of query buffers (ROWIDENT)

Returns: Nothing (procedure).

Notes: This will not fire if the field is changed manually in the fill-in, so you will need to use lookupleave as well to trap for this.

rowSelected (select.p)

Published from the browser when a row is selected.

Parameters: None

Returns: Nothing (procedure)

Notes: The browser is selected on default action and exits (if CancelrowObExit = false). The subscripting is done in browseHandler.

rowValues (data.p)

Retrieves a list of data from all rows in the DataObject.

Parameters:

`INPUT pcColumns AS CHARACTER`

Comma-separated list of RowObject column names.

`INPUT pcFormat AS CHARACTER`

`INPUT pcDelimiter AS CHARACTER`

Returns: CHARACTER

Notes:

- Intended for use by SmartSelect or other nonbrowser objects that need to show all rows of the SDO.
- Should not be used with large amounts of data since all data needs to fit in the return value.
- The purpose is to read all data without publishing to data sources. If the query is browsed, however, that happens when reposition-to-rowid is executed.

rowVisible (browser.p)

Determines if the records currently visible in the browse are the first or last records in the batch.

Parameters:

`INPUT pcRowids AS CHARACTER`

Comma-separated list of ROWIDs that are currently visible.

`INPUT phQryBuffer AS HANDLE`

Returns: CHARACTER

Notes: None

runInfo (toolbar.p)

Returns the necessary information for RUN or PROPERTY.

Parameters:

`INPUT pcAction AS CHARACTER`

Action Id where type = RUN.

`OUTPUT pocProcedure AS CHARACTER`

Target Procedure.

`OUTPUT pocParam AS CHARACTER`

Returns: Nothing (procedure)

Notes: Encapsulates the logic that looks in the parent for this information, if it is not defined in the action itself.

runServerObject (appserver.p)

Runs the server part of this object and sets AShandle.

Parameters:

`INPUT phAppService AS HANDLE`

AppServer Session handle.

Returns: Nothing (procedure)

Notes: Called from startServerObject and restartServerObject.

runServerProcedure (appserver.p)

Returns the handle of a procedure after it runs on server.

Parameters:

```
INPUT pcServerFileName AS CHARACTER

INPUT phAppService AS HANDLE
```

Returns: HANDLE

Notes: Simplifies an override of the run statement, such as to use a bind procedure instead of running the procedure directly.

saveContextAndDestroy (data.p)

Access: Saves the context of the server-side SmartDataObject in support of running in Stateless mode and then destroys it.

Parameters:

```
OUTPUT pcContext AS CHARACTER
```

The context string to be returned.

Returns: Nothing (procedure)

Notes: saveContextAndDestroy is invoked from the client side of the SmartDataObject and executed on the server side.

schemaField (b2b.p)

Returns the buffer-value of a schema field.

Parameters:

```
INPUT  phQuery AS HANDLE
```

Handle of a query defined for the schema.

```
INPUT  pcName  AS CHARACTER
```

Buffer name.

Returns: CHARACTER PRIVATE (buffer-value)

Notes: This is intended to be an interim solution.

searchTrigger (browser.p)

Contains the logic to support the SearchField property of the Dynamic SmartDataBrowser. The SmartDataBrowser repositions to the first record that is greater than or equal to the value entered in the SearchField.

Parameters: None

Returns: Nothing (procedure)

Notes: In the Data source connected to the SmartDataBrowser, searchTrigger Invokes in the following order the columnDataType, rowidWhere, and fetchRowIdent functions to reposition to a corresponding record.

selectFirstNode (treeview.p)

Selects the first node in the TreeView.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

selectNode (treeview.p)

Switches focus to node passed in by phNodeKey, selects this node, and generates a tvNodeSelected event.

Parameters:

```
INPUT pcNodeKey AS CHARACTER
```

Node to receive focus

Returns: LOGICAL

Notes: None.

selectPage (containr.p)

Changes the currently selected page. If the previous current page is not page 0 (the background page that is always visible), then hideObject is run in all the objects on the CurrentPage. Then the CurrentPage is changed to the new page number of piPageNum, and the changePage procedure is run to view and, if necessary, create the objects on the new page.

Parameters:

```
INPUT piPageNum AS INTEGER
```

Returns: Nothing (procedure)

Notes:

- This procedure is designed to switch from one page on a single frame to another, so the objects on the previous page are hidden and the objects on the new page viewed. If the new page is in fact a separate SmartWindow, then in most cases the similar procedure viewPage should be used, which will view the objects on the new page without hiding the current page.
- The selectPage procedure is run when a tab on the SmartFolder is pressed. It can also be run from application code when some other mechanism is used to change pages.

Examples:

```
/* This trigger code allows the user to select a page by typing the
   number into a field. */

ON LEAVE OF PageNum
DO:
   RUN SelectPage(INTEGER(PageNum:SCREEN-VALUE)).
END.
```

sendHandler (b2b.p)

Event handler for sending the message.

Parameters:

```
INPUT phMsgHandler AS HANDLE NO-UNDO.
```

Handle of message being sent.

Returns: Nothing (procedure)

Notes:

- Gets the actual schema and produces an XML document before calling super, which takes care of the actual saving of the document to the MEMPTR that is transferred to the message object.
- Overrides this to set header and other message information.

sendHandler (xml.p)

Sets the body of an outgoing message.

Parameters:

```
INPUT  phMessage AS HANDLE
```

Handle of the JMS message being sent.

Returns: Nothing (procedure)

Notes:

- The document handle is saved into a MEMPTR, and the MEMPTR is used to set the body of the Bytes Message.
- The DocumentHandle is expected to return an XML document.

sendMessage (b2b.p)

Overrides the msghandler message in order to set message-specific properties.

Parameters: None

Returns: Nothing (procedure)

Notes: None

sendMessage (msghandler.p)

Sends the message to the OutMessage-Target.

Parameters: None

Returns: Nothing (procedure)

Notes:

- This runs sendMessage with NO-ERROR and checks the error status after the call. If an error occurred, the errors are retrieved with fetchMessages() and passed to the sendErrorHandler. Errors that occur in sendMessage() in the OutMessageTarget are expected to be added to the message queue with addMessage().
- The Message Id returned from the Out Message Target will be stored in the CurrentMessageId property if ReplyRequired.

sendMessage (producer.p)

Creates and sends a message.

Parameters:

```
INPUT   pcDestination   AS CHARACTER

INPUT   plReplyRequired AS LOGICAL

INPUT   pcReplySelector AS CHARACTER

OUTPUT  pcMessageID     AS CHARACTER
```

Returns: Nothing (procedure)

Notes: This is run from the InMessage-Source to send a message.

SendMessageA (treeview.i)

Defined as EXTERNAL "user32"

Parameters:

```
INPUT hWnd AS LONG
```

The window handle

```
INPUT uMsg AS LONG
```

The message number

```
INPUT wParam AS LONG
```

The w param

```
INPUT lParam AS LONG
```

The l param

```
RETURN ReturnValue AS LONG
```

Returns: See the RETURN parameter (external procedure)

Notes: None.

sensitizeActions (panel.p)

Turns sensitivity on/off for some list of actions.

Parameters:

`INPUT pcActions AS CHARACTER`

The comma-separated list of actions to be altered

`INPUT plSensitive AS LOGICAL`

TRUE if the actions are to be made sensitive

Returns: LOGICAL

Notes: This is the static version; the dynamic version is in toolbar.p

sendReplyHandler (xml.p)

Processes a reply to a message.

Parameters:

`INPUT phMessage AS HANDLE`

Handle of the JMS reply message being sent.

Returns: Nothing (procedure)

Notes: An override of this procedure should set any necessary properties as well as supply the body of the reply message.

sendRows (data.p)

Fetches the requested number of rows from the database Query and creates corresponding records in the Row Object temp-table. The batch will typically start at row indicated by the pcRowIdent parameter, but can start on the row indicated by the piStartRow parameter.

Parameters:

`INPUT piStartRow AS INTEGER`

The RowNum to start on. An undefined value (?) indicates that the next argument (pcRowIdent) determines the start of the batch to be returned.

`INPUT pcRowIdent AS CHARACTER`

An alternative to StartRow. It is either "FIRST", "LAST", or a comma-delimited list of database ROWIDs. If it is "FIRST" or "LAST", the first or last batch of records in the data set is retrieved and piStartRow is forced to ?. If it is a comma-delimited list of database ROWIDs, the batch starts with a RowObject that comprises those records.

`INPUT plNext AS LOGICAL`

TRUE if the query should perform NEXT/PREV (depending on the direction of navigation) before starting to return more rows. In other words, "SKIP" to the next record at the start of the batch.

`INPUT piRowsToReturn AS INTEGER`

The maximum number of rows to return (supply a negative value to move backwards in the result set).

`OUTPUT piRowsReturned AS INTEGER`

The actual number of rows returned.

Returns: Nothing (procedure)

Notes: Before returning, sendRows repositions the RowObject query to what was the current row when it started. The pcRowIdent argument is used by fetchRowIdent to allow query repositioning to a specific database query row.

sensitizeActions (toolbar.p)

Set actions' SENSITIVE attribute. The main purpose is to have the same logic for disableActions and enableActions.

Parameters:

`INPUT pcActions AS CHARACTER`

Comma-separated list of the actions to be affected. "*" affects all actions.

`INPUT plSensitive AS LOGICAL`

TRUE if the actions are to be made sensitive.

Returns: LOGICAL PRIVATE

Notes:

- Actions can be created in other procedures, but this is the only place that they are enabled. The exception is that submenus are currently set TRUE when created.
- This procedure might need to be declared PUBLIC in order to call the canDo() function.

serverCommit (data.i)

Server-side counterpart of the client-side Commit function, used when running in n-tier (client + AppServer) mode. Receives from the client-side counterpart a dual set of records (original and changed versions) for update. If comparing the original and current versions reveals no outside changes, updates the records. Otherwise, returns the current (changed) versions to the client side without update, for display to the user.

Parameters:

`INPUT-OUTPUT TABLE FOR RowObjUpd`

The Update version of the RowObject temp-table.

`OUTPUT cMessages AS CHARACTER`

A CHR(3)-delimited string of accumulated messages from the server.

`OUTPUT cUndoIds`

List of any RowObject ROWIDs for which changes were rejected by a commit. Takes the form of RowNum/ADM-ERROR-STRING pairs, CHR(3) separators within pairs, comma separators between pairs.

Returns: Nothing (procedure)

Notes: The current versions of not-updatable records, if any, are returned in the RowObject temp-table.

serverCommit (datamessage.p)

Specialized message for data objects that gives the client the server API.

Parameters:

`INPUT-OUTPUT TABLE-HANDLE phRowObjUpd`

The update version of the RowObject temp-table

`OUTPUT pocMessages AS CHARACTER`

A chr(3)-delimited string of accumulated messages from the server

`OUTPUT pocUndoIds AS CHARACTER`

A list of any RowObject rowids whose changes need to be undone as the result of errors. The list takes the form:

RowNumCHR(3)ADM-ERROR-STRING,RowNumCHR(3)ADM-ERROR-STRING,...

Returns: Nothing (procedure).

Notes: None.

serverCommit (logic.i)

Not currently in use (commented out). Server-side update procedure.

Parameters:

Returns: Nothing (procedure).

Notes: None.

serverCommitTransaction (datamessage.p)

server-side version of CommitTransaction to receive all RowObjUpd table updates and pass them on to server-side SDOs.

Parameters:

`INPUT-OUTPUT TABLE-HANDLE phROwObjUpd1`

`...`

`INPUT-OUTPUT TABLE-HANDLE phROwObjUpd20`

Unused tables have dummy TABLE-HANDLE placeholders

Returns: Nothing (procedure).

Notes: None.

serverCommitTransaction (sbo.i)

Server-side version of CommitTransaction. Receives all RowObjUpd table updates and passes them on to server-side SDOs.

Parameters:

`INPUT-OUTPUT TABLE FOR...`

Up to 20 RowObjUpd table references, assembled through include files and macro definitions. Each table is dedicated to a contained SDO. If there are fewer than 20 contained SDOs, the unallocated tables are represented by dummy handles.

`OUTPUT pcMessages AS CHARACTER`

List of error messages.

`OUTPUT pcUndoIds AS CHARACTER`

List of ROWIDs for rows provoking error messages.

Returns: Nothing (procedure)

Notes: None

serverContainedSendRows (sbo.p)

Receives the SmartDataObject ObjectName and runs SendRows in that, returning the RowObject table. Server-side SmartBusinessObject version of serverSendRows.

Parameters:

`INPUT piStartRow AS INTEGER`

The RowNum value of the record to start the batch to return. Typically piStartRow is ? as a flag to use pcRowIdent instead of piStartRow.

`INPUT pcRowIdent AS CHARACTER`

The RowIdent of the first record of the batch to return. Can also be "FIRST" or "LAST" to force the retrieval of the first (or last) batch of RowObject records.

`INPUT plNext AS LOGICAL`

TRUE if serverSendRows() is to start on the record after the one indicated by piStartRow or piRowIdent.

`INPUT piRowsToReturn  AS INTEGER`

The number of rows in a batch.

`INPUT pcObjectName AS CHARACTER`

The ObjectName of the SmartDataObject to get data from.

`OUTPUT piRowsReturned AS INTEGER`

The actual number of rows returned. This number will either be the same as piRowsToReturn or less when there are not enough records to fill up the batch.

`OUTPUT TABLE-HANDLE phRowObject`

The batch of rows in the RowObject table.

Returns: Nothing (procedure)

Notes: None

serverFetchContainedData (datamessage.p)

The SBO version on the server-side will prepare and open a query and return all the resulting data to the client side.

Parameters:

```
INPUT pcQueries AS CHARACTER
```

CHR(1)-delimited-list of QueryString properties of the SDOs

```
INPUT pcPositions AS CHARACTER
```

reserved for future use to provide information on positioning each of the queries;

```
OUTPUT TABLE-HANDLE phRowObject1 AS HANDLE
                . . .
OUTPUT TABLE-HANDLE phRowObject20 AS HANDLE
```

One handle for each SDO's temp-table

Returns: Nothing (procedure).

Notes: None.

serverFetchContainedRows (datamessage.p)

Server-side procedure that retrieves a batch of data in one of the contained SDOs and returns all the resulting data to the client side.

Parameters:

`INPUT pcQueries AS CHARACTER`

CHR(1)-delimited-list of QueryString properties of the SDOs.

`INPUT piStartRow AS CHARACTER`

The object from which to retrieve the batch (? = Master)

`INPUT pcRowIdent AS CHARACTER`

The RowNum value of the record to start the batch to return. Typically piStartRow is "?" as a flag to use pcRowIdent instead of piStartRow.

`INPUT plNext AS CHARACTER`

True if serverSendRows is to start on the “next” record from what is indicated by piStartRow or piRowIdent.

`INPUT piRowsToReturn AS INTEGER`

The number of rows requested.

`OUTPUT piRowsReturned AS INTEGER`

The actual number of rows returned. This number will be less than the number requested if there are not enough additional records available.

```
OUTPUT TABLE-HANDLE phRowObject1 AS HANDLE
                . . .
OUTPUT TABLE-HANDLE phRowObject20 AS HANDLE
```

Temp-table handle for each SDO

Returns: Nothing (procedure).

Notes: None.

serverFetchContainedData (sbo.p)

Prepares and opens a query and returns all the resulting data to the client side. Server-side procedure.

Parameters:

```
INPUT   pcQueries      AS CHARACTER
```

CHR(1)-delimited list of QueryString properties of the SmartDataObjects.

```
INPUT   pcPositions    AS CHARACTER
```

Reserved for future use to provide information on positioning each of the queries.

```
OUTPUT  TABLE-HANDLE phRowObject1 . . .  phRowObject20
```

Temp-table handles for the maximum allowed number of SmartDataObjects.

Returns: Nothing (procedure)

Notes: None

serverFetchContainedRows (sbo.p)

Prepares and opens a query on the server side and returns all resulting data to the client side.

Parameters:

```
INPUT pcQueries AS CHARACTER
```

CHR(1)-delimited list of SDO QueryString properties.

```
INPUT pcPositions AS CHARACTER
```

Reserved for future use in providing positioning information for each of the queries

```
OUTPUT TABLE-HANDLE phRowObject1 ... phRowObject20
```

Temp-table handle for each SDO.

Returns: Nothing (procedure)

Notes: None

serverFetchDOProperties (sbo.p)

Runs at startup to return property values needed on the client. Server-side procedure.

Parameters:

```
OUTPUT pcPropList AS CHARACTER
```

Returns: Nothing (procedure)

Notes: Currently returns the OpenQuery property of each SmartDataObject.

serverFetchRowObjUpdTable (data.p)

Runs on the server side at startup to return property values needed by the client.

Parameters:

```
OUTPUT pcPropList AS CHARACTER
```

Properties and values in internal format that can be used for assigns.

Returns: Nothing (procedure)

Notes: Currently supplies the OpenQuery and IndexInformation property of each SDO. This routine performs its own initialization, so you can call it before initializeObject, if you choose.

serverSendRows (data.i)

Server-side routine called by its counterpart (clientSendRows) when operating in n-tier mode (client, AppServer). Acts as a pass-through proxy to sendRows for assembling and retrieving a batch of records. Runs in the ASHandle if in an SBO.

Parameters:

`INPUT  piStartRow AS INTEGER`

The RowNum value of the record to start the batch to return. Typically piStartRow is ? as a flag to use pcRowIdent instead of piStartRow.

`INPUT  pcRowIdent AS CHARACTER`

The RowIdent of the first record of the batch to return. Can also be "FIRST" or "LAST" to force the retrieval of the first (or last) batch of RowObject records.

`INPUT  plNext AS LOGICAL`

TRUE if serverSendRows is to start on the NEXT record offset from piStartRow/piRowIdent. Ignored if pcRowIdent is 'FIRST' or 'LAST'.

`INPUT  piRowsToReturn AS INTEGER`

The desired number of rows to return.

`OUTPUT piRowsReturned AS INTEGER`

The actual number of rows returned. This number will always be <= piRowsToReturn.

Returns: Nothing (procedure)

Notes:

- All of the parameters are simply received from the client and passed through to sendRows. The temp-table result is then received from sendRows and passed back to the client counterpart routine.
- If piStartRow is not 0 or ? then pcRowIdent is ignored. plNext is ignored if pcRowIdent is "FIRST" or "LAST".
- The most common use of piRowsReturned is to indicate that the entire result list has been returned when it is less than piRowToReturn.

serverSendRows (datamessage.p)

Specialized message for data objects that gives the client the server API.

Parameters:

`INPUT piStartRow AS INTEGER`

The RowNum value of the first record to return. Typically this will be "?", meaning: use pcRowIdent instead of piStartRow.

`INPUT pcRowIdent AS CHARACTER`

The RowIdent of the first record to return. Can also be "FIRST" or "LAST" to force the retrieval of the first (or last) batch of RowObject records.

`INPUT plNext AS LOGICAL`

True if serverSendRows is to start on the “next” record after piStartRow or piRowIdent.

`INPUT piRowsToReturn AS INTEGER`

The number of rows requested

`OUTPUT piRowsReturned AS INTEGER`

The actual number returned, which might be less than the number requested.

`OUTPUT TABLE-HANDLE phRowObject`

Returns: Nothing (procedure).

Notes: None.

ShellExecuteA (treeview.i)

Defined as EXTERNAL "shell32".

Parameters:

```
INPUT hWnd AS LONG
```

The window handle

```
INPUT lpOperation AS CHARACTER


INPUT lpFile AS CHARACTER


INPUT lpParameters AS CHARACTER


INPUT lpDirectory AS CHARACTER


INPUT nShowCmd AS LONG


RETURN hInstance AS LONG
```

Returns: Nothing (external procedure)

Notes: None.

showDataMessages (datavis.p)

Returns the name of the field (if any) from the first error message. Allows the caller to use it to position the cursor.

Parameters: None

Returns: CHARACTER

Notes: Invokes fetchMessages() to retrieve all Data-related messages (normally database update-related error messages), and displays them in an alert-box of type error. This function expects to receive back a single string from fetchMessages with one or more messages delimited by CHR(3), and within each message the message text, Fieldname (or blank) + a Tablename (or blank), delimited by CHR(4), if present.

showDataMessages (filter.p)

Returns the name of the field (if any) from the first error message to allow the caller to use it to position the cursor.

Parameters: None

Returns: CHARACTER

Notes: Invokes fetchMessages() to retrieve database update-related error messages), and displays them in an alert-box of type error. This function expects to receive back a single string from fetchMessages with one or more messages delimited by CHR(3), and within each message the message text, Fieldname (or blank) + a Tablename (or blank), delimited by CHR(4), if present.

showDataMessagesProcedure (datavis.p)

Replaces the showDataMessages function. Returns the name of the field (if any) from the first error message, to allow the caller to use it to position the cursor. A Dynamics procedure that uses Dynamics message handling routines.

Parameters:

```
OUTPUT pcReturn AS CHARACTER
```

Returns: Nothing (procedure).

Notes:

- Invokes fetchMessages() to retrieve all Data-related messages (normally database update-related error messages), and displays them in a alert-box of type error.
- This function expects to receive back a single string from fetchMessages with one or more messages delimited by CHR(3), and within each message the message text, Fieldname (or blank) + a Tablename (or blank), delimited by CHR(4) if present.

showMessage (smart.p)

Displays, using a simple MESSAGE statement by default, either a literal message string or the return value from a call to messageNumber.

Parameters:

```
INPUT pcMessage AS CHARACTER
```

A message string, which might be a message number in string form

Returns: LOGICAL

Notes: This function can be overridden to use a mechanism other than the MESSAGE statement to display messages, and still use the messageNumber function to map message numbers to translatable text. Note that this is different from addMessage, fetchMessages, etc., which log messages in a temp-table for later retrieval.

showMessageProcedure (smart.p)

Dynamics override for showMessage function to use Dynamics message handling routines. By default using a simple MESSAGE statement, displays either a literal message string, or a message number that is returned by the messageNumber function. The user's button choice is returned in the OUTPUT parameter.

Parameters:

`INPUT pcMessage AS CHARACTER`

Either a literal message string, or a message number in string form. A message number can be followed by a comma-separated list with up to 10 entries. All the entries except the last are used as replacements for parameters of form '&n' in the message string, where n is some number in the range 1 - 9. The last entry must be 'Question', 'YesNo', or 'YesNoCancel', which determines which buttons are provided in the message box.

`OUTPUT plAnswer AS LOGICAL`

The user's choice of button: true/false ('Question' or 'YesNo'), or true/false/undefined ('YesNoCancel')

Returns: Nothing (procedure).

Notes: None.

Examples:

```
showMessageProcedure ("Isn't this a nice &1 &2 message?", "short", "sample",
"Question").
```

showQuery (combo.p)

Displays the current query of the combo.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

showTVCError (tvcontroller.p)

Runs showMessages in the Session Manager to display a Treeview controller error message.

Parameters:

```
INPUT pcMessage AS CHARACTER
```

The message to be displayed

Returns: Nothing (procedure).

Notes: None.

showTVError (treeview.p)

Displays the error message passed as the argument.

Parameters:

```
INPUT pcMessage AS CHARACTER
```

The error message to display

Returns: Nothing (procedure).

Notes: None.

Signature (smart.p)

Returns the signature of the named function or internal procedure in the format returned by the Progress GET–SIGNATURE method.

Parameters:

```
INPUT pcName AS CHARACTER
```

The function or procedure name.

Returns: CHARACTER (Signature in Progress GET–SIGNATURE format.)

Notes: None

startSearch (browser.p)

Start-search trigger

Parameters:

```
INPUT phBrowse AS HANDLE
```

The browse handle.

Returns: Nothing (procedure).

Notes: Because persistent triggers have early binding, current-column cannot be passed in.

startServerObject (appserver.p)

Runs on the client to start the Object on the server.

Parameters: None.

Returns: Nothing (procedure).

Notes:

- restartServerObject will (should) be run the next time the object needs to communicate with its server part.
- Both start- and restart- call runServerObject, which issues the actual run call and does the optional initialization.
- This procedure is considered to be an internal event and returns without complaining if the object is bound or Appservice = ‘ ’. It does not check if ASHasStarted is false, as this may not be damaging if there’s no difference between start and restart. It is assumed to be called only when unbound and ASHasStarted is false as when called from getAsHandle or from initializeObject in objects that need to connect at start up.
- getASHandle depends for backward compatibility with data/sbo classes on the ‘silent error’ when Appservice is blank (not completely true, it actually checked ASDivision=‘client’, but that is too restrictive. So if errors are added here, make sure getAsHandle deals with (removes) them before returning.)

startServerObject (data.p)

Starts the SmartDataObject on the AppServer, when running statelessly. Runs on the client side.

Parameters: None.

Returns: Nothing (procedure).

Notes: This override is for error handling to show error message and return ‘adm-error’.

startServerObject (sbo.p)

Starts the server piece of a SmartBusinessObject. Called on the client.

Parameters: None.

Returns: Nothing (procedure).

Notes: This override is for error handling to show error messages and return ‘adm-error’.

start-super-proc (smart.i)

Starts a super procedure if it is not already running and adds it as a super procedure in any case.

Parameters:

```
INPUT pcProcName AS CHARACTER
```

Returns: Nothing (procedure)

Notes: None

startB2BObject (router.p)

Starts the Container with the SmartB2BObject, links it to the consumer, and returns its handle.

Parameters:

```
INPUT   pcContainer AS CHARACTER
```

Returns: HANDLE of the container

Notes: None

startDataRow (b2b.p)

Registers the dataRow record used to find, create, or update the data.

Parameters:

```
INPUT phDataSource AS HANDLE
```

DataSource handle.

```
INPUT pcAction AS CHARACTER
```

The operation to be carried out: “FIND”, “CREATE”, “UPDATE”, “CREATE,UPDATE” or “DELETE”.

Returns: LOGICAL

Notes: None

startElement (b2b.p)

Traps the event of a new XML element being processed, checks whether it is mapped to a dataObject, and, if so, registers it.

Parameters:

```
INPUT pcPath       AS CHARACTER
```

Logical path (not the numbered XPath).

```
INPUT pcNameSpace  AS CHARACTER
```

Namespace (future support).

```
INPUT pcName       AS CHARACTER
```

Name of the element.

```
INPUT pcQualName   AS CHARACTER
```

Name qualified with the namespace prefix.

Returns: Nothing (procedure)

Notes: None

startFilter (query.p)

Views/Starts the linked filter–source.

Parameters: None

Returns: Nothing (procedure)

Notes: None

startServerObject (appserver.p)

Runs on the client to start the Object on the server.

Parameters: None

Returns: Nothing (procedure).

Notes: None

startServerObject (data.p)

When a SmartDataObject is split and running statelessly on an AppServer, startServerObject is run on the client to start the SmartDataObject on the server.

Parameters: None

Returns: Nothing (procedure)

Notes: This override is for error handling to show error message and returns "adm–error".

startServerObject (sbo.p)

When a SmartBusinessObject is split and running statelessly on an AppServer, startServerObject is run on the client to start the SmartBusinessObject on the server.

Parameters: None

Returns: Nothing (procedure)

Notes: This override is for error handling to show error message and returns "adm–error".

startWaitFor (consumer.p)

Starts watching for messages when in batch mode.

Parameters: None

Returns: Nothing (procedure)

Notes: This is called from initializeObject when SESSION:BATCH-MODE = YES. It should not be called if the consumer is running in an interactive environment because it blocks user input.

stopHandler (consumer.p)

Clears the Waiting property to shut down the Consumer object.

Parameters:

```
INPUT  phMessage         AS HANDLE

INPUT  phMessageConsumer AS HANDLE

OUTPUT phReply           AS HANDLE
```

Returns: Nothing (procedure)

Notes: The stopHandler() routine is used to handle a ShutDown message, typically when in batch mode.

storeNodeValue (b2b.p)

Store node values for a data source until the endDocument event. (Consume XML)

Parameters:

`INPUT phDataSource AS HANDLE`

The data source

`INPUT pcColumnName AS CHARACTER`

The name of the node

`INPUT pcNodeValue AS CHARACTER`

The value

Returns: LOGICAL

Notes: None.

storeParameterNode (b2b.p)

Store mapping information for Output parameters (Produce XML)

Parameters:

`INPUT phDataSource AS HANDLE`

The data source

`INPUT pcMethod AS CHARACTER`

The mapping procedure/function

`INPUT pdNode AS DECIMAL`

The node bearing the value

`INPUT piNum AS INTEGER`

The parameter number

```
INPUT piNumParam AS INTEGER
```

The total number of parameters in the method

Returns: LOGICAL

Notes: This information is stored until the parent node of the first parameter is processed. callOutParams() will then call the pcmethod in phDataSource and use the piNum parameter values to populate the node or attribute.

storeParameterValue (b2b.p)

Stores values for input parameters.

Parameters:

```
INPUT  phDataSource AS HANDLE
```

DataSource.

```
INPUT  pcMethod AS CHARACTER
```

Procedure or function.

```
INPUT  piNum AS INTEGER
```

Parameter number.

```
INPUT  piNumParam AS INTEGER
```

Number of parameters in the method.

```
INPUT  pcValue AS CHARACTER
```

Value to pass as input.

Returns: LOGICAL

Notes: This information is stored until the endDocument call. InParams calls the pcmethod in phDataSource and passes the PcValue as the piNum parameter.

storePendingSensitivity (toolbar.p)

To store pending sensitivity settings used to resolve timing issues where sensitivity is set before the toolbar is built.

Parameters:

```
INPUT pcActions AS CHARACTER


INPUT plSensitive AS LOGICAL
```

Returns: Nothing (procedure).

Notes: None.

stripCalcs (browser.p)

Strips the expression portion of expression @ variable

Parameters:

```
INPUT cClause AS CHARACTER
```

The clause to be edited

Returns: CHARACTER

Notes: None.

submitCommit (data.p)

Called by submitRow. Commits changes to the database.

Parameters:

`INPUT pcRowIdent`

The identifier of the row to commit. It is a comma-delimited list of ROWIDS. The first entry is the ROWID of the RowObject temp-table containing the record to commit. Subsequent entries in the comma-delimited list are the database ROWIDs of the records to be modified.

`INPUT plReopen`

TRUE if the RowObject query is to be reopened and repositioned to the RowObject record identified by pcRowIdent.

Returns: Nothing (procedure)

Notes: None

submitForeignKey (data.p)

Called from submitRow. Defines Foreign Key values for a new row.

Parameters:

`INPUT pcRowIdent`

The RowIdent (a comma-delimited list of ROWIDs that uniquely identify the RowObject record) of the new RowObject record to be stored in the database. Since this is a new record, only the first ROWID (that of the RowObject itself) is valid. There are no database ROWIDs because they have not yet been created.

`INPUT-OUTPUT pcValueList`

A CHR(1)-delimited list of column and value pairs to be set in the RowObject record identified by pcRowIdent. On input it is a list that has been set before submitForeignKey was called. On output it is the now current list with the addition of any foreign fields that have been set.

`INPUT-OUTPUT pcUpdColumns`

A comma-delimited list of column names that are updatable in the SmartDataObject. If any ForeignFields are set and added to the pcValueList, they will also be added to pcUpdColumns to let the calling code know that it is permissible to update them.

Returns: Nothing (procedure)

Notes:

- The list of updatable fields (pcUpdColumns—derived from the UpdatableColumns property in submitRow) usually does not contain key fields. However, because submitForeignKey is called when creating a new RowObject record, key fields need to be populated, so pcUpdColumns is expanded to allow for this. However, only the variable pcUpdColumns is expanded. The UpdatableColumns property remains unchanged.

- This procedure allows foreign key values to be assigned automatically when new records are created. For example, if the current SmartDataObject is for Orders of the current Customer, where the Customer is maintained in a parent SmartDataObject query, then when new orders are added, the CustNum value for the current Customer should be assigned automatically to all newly created Orders. This procedure does that by adding the CustNum field and its value to the list of modified fields. In addition, since these ForeignFields are often not directly updatable (would not be enabled in a visualization), the code also needs to let the calling code (the submitRow function) know that the Foreign Field update should be allowed. It is for this reason that the UpdColumns parameter is passed; the key fields being updated will be added to the parameter value if they are not already there. As noted, this will modify the list of updatable columns only for this one transaction; it does not cause the caller to modify the UpdatableColumns property itself.

- The submitForeignField procedure can be localized in a particular application's SmartDataObject to make other changes to the initial values of newly created records by modifying the two INPUT–OUTPUT parameters as described here; what is done in the standard code for ForeignFields can be done by application code for other fields to be initialized.

submitRow (data.p)

Accepts a list of changed values for a row and ASSIGNs them, returning FALSE if any errors occur. This is done only to the RowObject temp-table. Committing the changes back to the database is a separate step, which will be invoked from here if AutoCommit is set on.

Parameters:

`INPUT pcRowIdent AS CHARACTER`

The "key" with row number to update, plus a list of the ROWID(s) of the database record(s) from which the RowObject is derived.

`INPUT pcValueList AS CHARACTER`

A CHR(1)-delimited list of alternating column names and values to be assigned.

Returns: LOGICAL

Notes: None

submitRow (sbo.p)

Accepts a list of changed values for a row and passes them on to the SmartDataObjects from which they came.

Parameters:

`INPUT  pcRowIdent AS CHARACTER`

"Key" with row number to update, plus a list of the ROWIDs of the db records from which the RowObject is derived.

`INPUT  pcValueList  AS CHARACTER`

A CHR(1)-delimited list of alternating column names and values to be assigned.

Returns: LOGICAL

Notes: None

submitValidation (data.p)

Performs validation on each of the columns of a new RowObject record, and on the record as a whole, before committing it to the database.

Parameters:

`INPUT pcValueList AS CHARACTER`

A CHR(1)-delimited list of Column Name<->value pairs that need validation before committing a transaction.

`INPUT pcUpdColumns AS CHARACTER`

A comma-delimited list of the SmartDataObject columns that are updatable.

Returns: Nothing (procedure)

Notes: There are four types of validation performed:

- Newly entered data meets the format criteria.
- Newly entered data is in an updatable field.
- Any developer-defined validation in the form of a *fieldname*Validation procedure.
- Any developer-defined validation in the form of a RowObjectValidation procedure for the whole row.

substituteActionText (toolbar.p)

Substitutes &1 in the passed text with the substitute property.

Parameters:

`INPUT pcAction AS CHARACTER`

The action

`INPUT pcText AS CHARACTER`

The text to substitute in

Returns: CHARACTER

Notes: Used by actionLabel, actionCaption, and actionTooltip overrrides

supportedObjects (toolbar.p)

Returns a comma separated list of object names for object linked with supported links. Semi-colon is used to separate a potential RunAtttribute

Parameters:

```
INPUT plLoaded AS LOGICAL
```

FALSE if only unloaded objects are to be included

Returns: CHARACTER

Notes: The Container's RunAttribute

synchronizeProperties (data.p)

Synchronizes certain properties on the client side of a SmartDataObject with its server side. This is used internally to support running SmartDataObjects in stateless mode.

Parameters:

```
INPUT pcPropertiesForServer
```

A CHR(3)-delimited list of *prop* CHR(4) *value* pairs to be set on the server half of the SmartDataObject.

```
OUTPUT pcPropertiesForClient
```

A CHR(3)-delimited list of "prop CHR(4) value" pairs to be set on the client half of the SmartDataObject.

Returns: Nothing (procedure)

Notes:

- synchronizeProperties is designed to be invoked from the client side but executed on the server half of a SmartDataObject. While executing on the server side it calls setPropertiesList to set the properties in pcPropertiesForServer, then accumulates properties to be sent back to the client so that the client can set them. Currently, the only property sent back is the ServerOperatingMode property.
- The form of both pcPropertiesForServer and pcPropertiesForClient is designed to work with the setPropertyList method.

tableOut (data.p)

Output requested fields of SDO in standard temp-table

Parameters:

`INPUT pcFieldList AS CHARACTER`

The list of fields wanted

`INPUT plIncludeObj AS LOGICAL`

Whether to include object fields

`INPUT piMaxRecords AS INTEGER`

Maximum number of records to handle (currently must be <= 5000)

OUTPUT TABLE FOR ttTable

The SDO data

`OUTPUT iExtractedRecs AS INTEGER`

The actual number of records handled

Returns: Nothing (procedure).

Notes:

- Temp table is defined in afttsdoout.i. Fields passed in are checked with a can-do to support "*" for "all" or "! field_name" to exclude a field, e.g. " ! RowNum,!RowIdent,!RowMod, * " would use all non-SDO specific fields. Do not use a table prefix for fields passed in. The temp table contains a record per record / field combination Currently has a hard coded limit of 5000 records to fetch - max!

targetActions (panel.p)

Return the actions of a target

Parameters:

INPUT pcLinkType AS CHARACTER

The type of target

Returns: CHARACTER

Notes: This is the static version used for static buttons in panels. Overridden in toolbar

targetActions (toolbar.p)

Returns the actions that apply to a specific target.

Parameters:

```
INPUT pcLinkType AS CHARACTER
```

The type of target

Returns: CHARACTER

Notes: None.

targetPage (containr.p)

Returns the ADM page number associated with some object.

Parameters:

INPUT phObject AS HANDLE

The object of interest

Returns: INTEGER

Notes: None.

targetProcedure (b2b.p)

The Router subscribes this to an event of the schema name in order to find whether it is already started.

Parameters:

```
OUTPUT phHandle AS HANDLE.
```

phHandle = TARGET-PROCEDURE.

Returns: Nothing (procedure)

Notes: None

toggleData (smart.p)

Uses the {set} pseudo-function to set a new value for DataLinksEnabled.

Parameters:

```
INPUT plEnabled AS LOGICAL
```

Returns: Nothing (procedure).

Notes: None.

toolbar (browser.p)

Carries out one of several operations depending on the value of the argument passed in. Valid values are: ‘comments’; ‘export’ for exporting data to Excel; ‘preview’ for previewing a report; ‘filter’ or ‘find’ to launch a filter window; ‘view’, ‘copy’, ‘modify’, or ‘add’ to launch the appropriate folder window; ‘delete’ to delete a record with autocommit; and ‘audit’.

Parameters:

```
INPUT pcValue AS CHARACTER
```

The type of operation

Returns: Nothing (procedure).

Notes: None.

toolbar (containr.p)

Generic event handler for toolbar events

Parameters:

```
INPUT pcValue AS CHARACTER
```

The string used by the handler's CASE statement to determine behavior. Valid values are:

- EnableData — Activates data links by publishing ToggleData.
- DisableData — Deactivates data links by publishing ToggleData.
- Notepad — Launches the Notepad application by calling launchExternalProcess.
- Wordpad — Launches the Wordpad application by calling launchExternalProcess.
- Calculator — Launches the Calculator application by calling launchExternalProcess.
- Internet — Launches the Internet Explorer browser by calling launchExternalProcess.
- Email — Runs sendEmail.
- Word — Launches the Word word-processor application.
- Excel — Launches the Excel spreadsheet application.
- PrintSetup — Opens the system's printer-setup dialog box.
- Suspend — Runs af/cod2/aftemsuspd.w
- Re-Logon — Runs relogon.
- Preferences — Launches the preferences applet rydynpref.w.
- Translate — Launches the translation applet rydyntran.w.
- Help — Runs contextHelp.
- HelpAbout — Runs helpabout.
- HelpTopics — Runs helptopics.
- HelpContents — Runs helpcontents.
- HelpHelp — Runs helphelp.
- Spell, Audit, Comments, History, Status — Currently trapped, but undefined.

Returns: Nothing (procedure).

Notes: None.

toolbar (viewer.p)

Toolbar event handler. Handles the cases ‘OK’ and ‘Cancel’.

Parameters:

`INPUT pcValue AS CHARACTER`

Valid values: ‘OK’, ‘Cancel’

Returns: Nothing (procedure).

Notes: None.

transferDBRow (query.p)

Buffer–copies the database records for the current query row into the RowObject temp-table.

Parameters:

`INPUT pcRowIdent AS CHARACTER`

A comma-separated list consisting of the ROWID of the RowObject record followed by the RowIds of the database records; if unknown or blank, then the database records have already been retrieved.

`INPUT piRowNum AS INTEGER`

The integer value to assign to the RowNum field in the RowObject temp-table record.

Returns: Nothing (procedure)

Notes: This procedure replaces the transferRowFromDB function from Version 9.0. It can be localized to perform additional custom operations each time a row is transferred from the database to the RowObject table.

transferToExcel (data.p)

Transfers the contents of the SDO to Excel

Parameters:

`INPUT pcFieldList AS CHARACTER`

The list of fields to be transferred

`INPUT plIncludeObj AS LOGICAL`

Whether to include object fields

`INPUT plUseExisting AS LOGICAL`

Whether to use the currently-running copy of Excel

`INPUT piMaxRecords AS INTEGER`

Maximum number of records to transfer

Returns: Nothing (procedure).

Notes: Always excludes rowobject specific fields, e.g. RowNum, RowIdent, RowMod. Uses tableout procedure defined in here also.

translateBrowseColumns (lookup.p)

Translates lookup and filter browse columns.

Parameters:

`INPUT pcObjectName AS CHARACTER`

`INPUT phBrowseHandle AS HANDLE`

Returns: Nothing (procedure).

Notes: None.

unbindServer (appserver.p)

Unbinds the AppServer by destroying the server side object started by the client.

Parameters:

```
INPUT pcMode AS CHARACTER
```

Valid values are ‘unconditional’, ‘conditional’, and the empty string. If ‘conditional’, unbinding will only take place if the caller is at the same level as BindSignature. The empty string is recognized, but not currently supported.

Returns: Nothing (procedure).

Notes:

- This procedure allows nested calls of procedures that binds and unbinds, but still postpone the unbinding until we are back at the level that did the actual binding.
- The logic in this procedure is dependent of the fact that getAsHandle or bindServer does the actual binding (if AsHandle is ?) and logs the call level by setBindSignature = program-name(2).
- An external caller will use the following sequence to ensure that all calls are done with one connection:

```
RUN bindServer in <handle>.
somerequest in handle
somerequest in handle
RUN unbindServer(?).
```

- Internal calls will typically look like this;

```
hAsHandle = getAsHandle().
somerequest in hasHandle
RUN unbindServer(?).
```

- Overrides need to do the following:

```
def input param pcmode as char.
run SUPER(if pcMode = ? then program-name(2) else pcMode).
```

- Limitations apply as recursive calls and external callers have the same signature, so an unbind might happen too early. (This can probably be fixed by saving the complete stack in BindSignature)

unBlankFillin (filter.p)

Resets the format of a blank fill-in.

Parameters:

`INPUT phField`

The handle of a field.

Returns: LOGICAL

Notes: The blankFields changes the format of fill-ins to make it appear blank. This function is called as soon as the user starts editing the field; VALUE-CHANGED - > "U10" triggers in `filter.i` makes this happen.

unBlankLogical (filter.p)

Resets the format of a blank fill-in of data-type LOGICAL.

Parameters:

`INPUT phField AS HANDLE`

INPUT phField - the handle of a field

Returns: Nothing (procedure)

Notes: The blankFields changes the format of fill-ins to make it appear blank. This procedure is defined ON ANY-PRINTABLE PERSISTENT trigger when the field is blanked.

undoClientUpdate (data.i)

Rolls back updates to the RowObject table in the event of a client-side error. Using a FOR EACH loop, BUFFER-COPYs appropriate RowObjUpd records to RowObject, omitting RowMod.

Parameters: None.

Returns: Nothing (procedure).

Notes: This is to deal with errors before commit has been run - i.e. client side validation errors.

undoTransaction (data.p)

Undoes any uncommitted changes to the RowObject table when the "Undo" button is pressed in the commit panel.

Parameters: None

Returns: Nothing (procedure)

Notes: The undoTransaction calls doUndoTrans to restore the RowObject temp-table and empty the RowObjUpd temp-table.

undoTransaction (sbo.p)

Passes the undoTransaction event on to each contained SmartDataObject that has any uncommitted changes.

Parameters: None

Returns: Nothing (procedure)

Notes: None

updateActive (containr.p)

Published from ContainerTargets when they change state as a result of, for example, setDataModified, setNewRecord, or setRowObjectState.

Parameters:

```
INPUT plActive AS LOGICAL
```

TRUE if some update is known active and ContainerTargets need not be surveyed

Returns: Nothing (procedure).

Notes: This is part of the logic to make the UpdateActive property reflect the containers state.

updateActive (toolbar.p)

Published from Toolbar-Target (the container) to signal that some of its objects have changed state. Runs resetToolbar in TARGET-PROCEDURE.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

updateAddQueryWhere (data.p)

Restores a query to design-time state, re-adds the filters, then adds and saves the new WHERE clause.

Parameters:

`INPUT pcWhere AS CHARACTER`

New WHERE clause

`INPUT pcField AS CHARACTER`

Field to search and replace

Returns: Nothing (procedure).

Notes:

- Ensures WHERE clause added is not cleared by filters, and is not duplicated in saved manualaddquerywhere. Where clause must be in correct format for an addquerywhere. If WHERE clause is blank but the field is passed in, then any manual query for that field will be removed, therefore defaulting back to all.
- Only supports adding a single where clause, but may be called many times. Used when putting a manual filter viewer above a browser!

updateCategoryLists (toolbar.p)

Updates the lists of available categories for the toolbar. Called from loadBands at design time. The result is stored in AvailMenuActions and AvailToolbarActions.

Parameters:

`INPUT pcBand AS CHARACTER`

The band

`INPUT pcTopLevelType AS CHARACTER`

The type of the band directly connected to the toolbar

`INPUT-OUTPUT pcMenuCategories AS CHARACTER`

The list of categories

```
INPUT-OUTPUT pcToolbarCategories AS CHARACTER
```

The list of categories

Returns: Nothing (procedure).

Notes: None.

updateManualForeignFields (data.p)

Get foreign fields and values and setManualAssignQuerySelection so that filter does not lose info

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

updateMode (datavis.p)

Enables/disables fields when the data visualization object is linked to an Update SmartPanel for which the type is “Update”.

Parameters:

```
pcMode AS CHARACTER
```

Update mode possible values are:

- **updateBegin** – Signals that the user has pressed the update button.
- **updateEnd** – Signals that the user has pressed the save button.

Returns: Nothing (procedure)

Notes: “UpdateBegin” signals that the Update button has been pressed in an Update SmartPanel in Update mode. Enable fields for data entry. ' updateEnd' signals that Save has completed; disable fields. ' update-begin' was a state msg in V8. updateEnd corresponds to the save button dispatching disable-fields in V8.

updateQueryPosition (data.p)

Resets the QueryPosition property after a record navigation. The main purpose of this function is to eliminate duplication and errors and minimize messaging (setQueryPosition PUBLISHes) in fetchFirst, fetchPrev, fetchNext, and fetchLast.

Parameters: None

Returns: Nothing (procedure)

Notes:

- `data.p` should update LastRowNum, FirstRowNum, and LastDbRowIdent properties and call the reset function.
- The properties LastRowNum and FirstRowNum stores the RowObject.RowNum of the first and last record in the database query.
- The LastDbRowIdent is updated at openQuery() if CheckLastOnOpen is TRUE and enables identification of the last record in cases where LastRowNum has not yet been updated.

updateRecord (browser.p)

SmartDataBrowser version of updateRecord.

Initiates the “Save” process after an add, copy, or update. This version of updateRecord stores the RowIdent property of the current record if it is an add or a copy.

Parameters: None

Returns: Nothing (procedure)

Notes:

- Invokes updateRecord in `datavis.p` (RUN SUPER).
- If an error occurs after invoking updateRecord in `datavis.p` (which means the update failed), updateRecord puts focus back in the SmartDataBrowser if it is not already there.
- If the SmartDataBrowser has its own database query, and if the save operation comes after an add or a copy, the SmartDataBrowser’s query is reopened and repositioned to the last row of the viewport.

updateRecord (datavis.p)

Receives a “Save” message from a TableIO source and sends the changed values back to the SmartDataObject. General code to start the save operation.

Parameters: None

Returns: Nothing (procedure)

Notes:

Invoked when the Save button is pressed on an Update SmartPanel or a SmartToolbar. PUBLISHes “collectChanges” to gather changes made eventually in any GroupAssign target. Invokes submitRow() in the Data source that actually applies the changes to the database. If any validations failed or errors occurred during the save operation for a field, it brings up the page where the field is, if necessary, and repositions to it.

updateRow (data.p)

Updates an existing row.

Parameters:

```
INPUT pcKeyValues AS CHARACTER
```

pcKeyValues – comma-separated or CHR(1)-separated list of keyvalues

```
INPUT pcValueList AS CHARACTER
```

row pcValueList – CHR(1) delimited list of alternating column names and values to be assigned.

Returns: LOGICAL

Notes: None

updateState (browser.p)

Receives “state” messages on updates. Enables or Disables the SmartDataBrowser while updates are made to other objects sharing the same SmartDataObject. SmartDataBrowser version of updateState.

Parameters:

```
INPUT PcState AS CHARACTER
```

The update state value to be passed to the SmartDataBrowser. Valid values are "UpdateBegin", "Update", and "UpdateComplete".

Returns: Nothing (procedure)

Notes:

- Possible input values are UpdateBegin, Update, and UpdateComplete.
- Returns if the DataModified property equals TRUE (meaning updates are in progress), or if the NewRecord property is not equivalent to “No” (which means add or copy).
- Used internally.

updateState (data.p)

Passes along update-related messages (to its Navigation–Source, for example.) and adjusts the DataModified property.

Parameters:

```
INPUT pcState AS CHARACTER
```

An indicator of the state of an updatable record. Valid values are:

- **"UpdateBegin"** — The user has indicated that an update will take place, either by pressing the Update button in a panel or by entering an updatable field in a browser.
- **"Update"** — An update is in progress in another object (a viewer or browser).
- **"UpdateEnd"** — A save has completed (the same as "UpdateComplete".)
- **"UpdateComplete"** — Any changes to the RowObject temp-table have been either committed or backed out.

Returns: Nothing (procedure)

Notes: The SmartDataObject also saves its own “copy” of the DataModified property, set TRUE when updateState is "Update" and FALSE when updateState is "UpdateComplete", so that it can be queried by other objects (such as other Data–Targets).

updateState (datavis.p)

Receives the updateState event (for example, from a GroupAssign target) and sets its own DataModified property accordingly (which in turn passes on the updateState event).

Parameters:

```
INPUT pcState AS CHARACTER
```

pcState AS CHARACTER update state Possible values are:

- **Update** – Signals that update is in progress.
- **Complete** – Signals that update is terminated.

Returns: Nothing (procedure).

Notes: None

updateState (panel.p)

Receives state message events related to record updates.

Parameters:

```
INPUT pcState AS CHARACTER
```

Update state can be "Update" or "UpdateComplete"

Returns: Nothing (procedure)

Notes:

- For "Update", a Navigation Panel will set its PanelState property to disable–all, to prevent navigation during an update operation. A Save or Update panel will set its state to "action–chosen", meaning that a button has been pressed that puts the panel in the middle of an Update or Add or Copy operation that must be explicitly Saved to be completed.
- For "UpdateComplete", a Navigation Panel will restore the prior PanelState (having been disabled when the update began). A Save or Update panel will set its state to ‘add–only’ if no rows are available in the current dataset, to "disable–all" if no rows are available and there is also no available parent row for this dataset (meaning that not even an Add could be done), or to "initial" if the RecordState is RecordAvailable, meaning that all operations are allowed (Update/Add/Copy/Delete).
- In addition, for an Update panel, which explicitly enables fields when an Update begins and disables them when the operation is Saved, the updateMode event is published with a value of "UpdateEnd", to tell visualizations to disable their fields.

updateState (sbo.p)

Republishes any updateState event messages received from a Data-Target, to get them, for example, to Navigation Panel/Toolbar.

Parameters:

```
INPUT pcState AS CHARACTER
```

Returns: Nothing (procedure)

Notes: None

updateState (toolbar.p)

Receives state message events related to record updates.

Parameters:

```
INPUT pcState AS CHARACTER
```

Update state.

Returns: Nothing (procedure)

Notes: None

updateState (tvcontroller.p)

Disables the Treeview during user editing and re-enables on completion.

Parameters:

```
INPUT pcState AS CHARACTER
```

The new state.

Returns: Nothing (procedure).

Notes: None.

updateState (viewer.p)

Initialization code specific to SmartDataViewers.

Parameters: None

Returns: Nothing (procedure)

Notes:

- initializeObject reads through the list of enabled and displayed fields and gets their handles to store in the FieldHandles and EnabledHandles properties.
- dataAvailable is invoked to check to see if the data-source already has a record available for display.

If there is an Update–Target and a TableIO–Source in Save mode, or if this is part of a GroupAssign and the GroupAssign–Source is already enabled, enableFields is invoked to enable fields in the ENABLED–FIELDS list.

updateStates (toolbar.p)

Updates state temp-table whenever bands/actions are loaded. Adds actions to already-loaded states.

Parameters:

```
INPUT pcState AS CHARACTER
```

The name of some state. Empty string means ‘all’.

Returns: Nothing (procedure).

Notes: None.

updateTitle (browser.p)

SmartDataBrowser version of updateTitle.

Parameters: None.

Returns: Nothing (procedure).

Notes: Setting up a window title for an object controller is different than for a viewer. This code gets the information from the browser’s data-sources.

updateTitle (datavis.p)

Updates window title at end of update.

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

updateTree (treeview.p)

Updates details of nodes that are currently in the tree.

Parameters:

```
INPUT phTable AS HANDLE
```

Handle of tTreeData

Returns: Nothing (procedure).

Notes: None.

validateBuffer (action.p)

Validates the data in the actions buffer.

Parameters:

```
INPUT pcBuffer AS CHARACTER

INPUT pcKey AS CHARACTER
```

Returns: LOGICAL

Notes: This routine is called after the create operation.

validateFields (datavis.p)

Validates a field in a data-display object.

Parameters:

```
INPUT-OUTPUT pcNotValidFields AS CHARACTER
```

Valid values on output are:

- The empty string, if validation succeeded
- A comma-separated list of the fieldname, object handle and page, if validation failed

The updateRecord routine will use this output value to set focus.

Returns: Nothing (procedure)

Notes:

- Called from updateRecord to ensure that data can be saved.
- Field validation on leave cannot be trusted because save can be triggered by accelerators and no-focus buttons (toolbar).
- Currently, only one error is handled. Validate() on hidden objects does not give any message.
- Validation will always check unmodifed fields. This is useful for add/copy but not otherwise, as those fields will not be saved.

valueChanged (combo.p)

Makes sure the SmartDataViewer knows that the Combo's value has changed. Publishes populateChildCombo and comboValueChanged.

Parameters: None.

Returns: Nothing (procedure).

Notes: Defined as a PERSISTENT value-changed trigger in the dynamic-widget. The code currently duplicates the logic that is defined in the U10 trigger in the SmartDataViewer because this code's check of FOCUS is wrong when the actual change is caused by a user event in the browser.

valueChanged (lookup.p)

Uses the {set} pseudo-function to set DataModified to YES. This ensures that the SmartDataViewer container knows that the Lookup's value has changed.

Parameters: None.

Returns: Nothing (procedure).

Notes: Defined as a PERSISTENT value-changed trigger in the dynamic-widget. The code currently duplicates the logic defined in the U10 trigger in the SmartDataViewer because checking FOCUS is not helpful when the change is caused by a user event in the Browser.

valueChanged (select.p)

Makes sure the SmartDataViewer container knows that this value has changed.

Parameters: None

Returns: Nothing (procedure)

Notes: Defined as a PERSISTENT value-changed trigger in the dynamic-widget. The code currently duplicates the logic that is defined in the U10 trigger in the SmartDataViewer when the view-as is browse, because this code's check of FOCUS is wrong when the actual change is caused by a user event in the browser.

valueChanged (viewer.p)

Value-changed of frame anywhere event

Parameters: None.

Returns: Nothing (procedure).

Notes: Static objects can reach this also by applying U10 event to run it.

viewHideActions (panel.p)

Views and hides static actions

Parameters:

`INPUT pcViewActions AS CHARACTER`

The comma-separated list of actions to make visible

`INPUT pcHideActions AS CHARACTER`

The comma-separated list of actions to make hidden

Returns: Nothing (procedure).

Notes: Dynamic version in toolbar.p

viewHideActions (toolbar.p)

Sets actions visible/hidden according to state (called from setbuttons). Loops through the pcViewActions list, then the pcHideActions.

Parameters:

```
INPUT pcViewActions AS CHARACTER
```

List of actions to be made visible

```
INPUT pcHideActions AS CHARACTER
```

List of actions to be made HIDDEN

Returns: Nothing (procedure).

Notes: For buttons we simply hide/view existing buttons and then remove them into new positions. For menu items we must reconstruct any submenus that have been modified as menu items do not have a hidden attribute.

viewObject (browser.p)

Overrides run setDown to set the DOWN attribute for the browse when it is being viewed and to fix problems caused by the browse's implicit visualization behavior.

Parameters: None

Returns: Nothing (procedure)

Notes:

- setDown must be called from viewObject as opposed to initializeObject to support the object being on a page other than zero, because set down requires a visible browse. This sets DOWN when the object is actually being viewed rather than when it is initialized, which could happen before it is viewed if it is on a page other than zero.
- The browser implicitly sets the buffer to its currently selected row when it is visualized. This might cause problems if the DataSource is being manipulated by another object. An update-target might have called AddRow in the DataSource, which creates a RowObject while the browse still has the old row selected.

viewObject (combo.p)

Make sure the label is visible by turning off HIDDEN.

Parameters: None.

Returns: Nothing (procedure).

Notes: For sizing reasons, the label is not really a part of the object, but added as a text widget to the parent frame.

viewObject (containr.p)

Container-specific code for viewObject. If the HideOnInit property has been set during initialization to allow this object and its contents to be initialized without being viewed, turn that off here and explicitly view all contents.

Parameters: None

Returns: Nothing (procedure)

Notes:

- This Container-specific version of viewObject checks the HideOnInit property for the Container. If it was TRUE, then the container was not viewed when first initialized. Now that it is being viewed, it turns off the HideOnInit property in itself and its contents, and explicitly visualizes its contents.
- See viewObject for `visual.p` for more general information on using and customizing this event procedure.

viewObject (lookup.p)

Make sure the label is not HIDDEN.

Parameters: None.

Returns: Nothing (procedure).

Notes: For sizing reasons, the label is not really a part of the object, but added as a text widget to the parent frame.

viewObject (select.p)

Makes sure the label is viewed.

Parameters: None

Returns: Nothing (procedure)

Notes: None

viewObject (smart.p)

Logically “views” the current object, setting its ObjectHidden property to false.

Parameters: None

Returns: Nothing (procedure)

Notes:

- The ADM supports a logical concept of “Viewing”, which can be applied to all objects regardless of whether they have a visualization; that is why this procedure is found in `smart.p`.
- When an object is “viewed” its links are typically activated (the linkState property is set to "active"); when “hidden” they are sometimes deactivated, depending on the link type. If an object has an actual visualization, the version of viewObject in `visual.p` will view it.

- When a SmartContainer is viewed, it is not necessary for all of the objects it contains to be individually viewed, because they are not explicitly hidden when the container is hidden (see hideObject). For this reason, viewObject, when executed for a SmartContainer, will set the property ContainerHidden to FALSE in each contained SmartObject, without actually running viewObject in each object. The setContainerHidden function will, in turn, set the ObjectHidden property to FLASE so that it can be queried successfully. Not running hideObject and viewObject in each individual SmartObject improves performance when SmartObjects are being paged (alternately hidden and viewed), and can eliminate problems with "flashing" of visual objects or problems restoring proper frame order when objects are hidden and viewed.

Examples:

```
/* Views an object when a button is pressed. */

ON CHOOSE OF Btn_View_Browser
DO:
  RUN viewObject IN hSDBrowser.
END.
```

viewObject (toolbar.p)

Views the object. The purpose for the override is to add the menu bar handle to the window.

Parameters: None

Returns: Nothing (procedure)

Notes: None

viewPage (containr.p)

Views a new page without hiding the current page. This is intended to be run from application code that wants to view a new page that is a separate SmartWindow. viewPage will run changePage to view (and if necessary create) the new SmartWindow, but will not hide the objects on the current page, since they are in a separate window that can be viewed at the same time.

Parameters:

```
INPUT piPageNum AS INTEGER
```

Returns: Nothing (procedure)

Notes:

- Because the previous page is not hidden, the CurrentPage property is reset only temporarily so that changePage knows the new page number; then it is reset to its previous value.
- Use the procedure selectPage to hide the current page in a container and view a different one.

Examples:

```
/* In this example, a SmartWindow has been placed onto page 5 of a
   containing SmartWindow. When a button is pressed, viewPage views the
   subwindow without hiding anything or changing the current page on the
   main SmartWindow. */

ON CHOOSE OF Btn_SubWin
DO:
  RUN viewPage(5).
END.
```

viewRecord (viewer.p)

Set object in viewMode

Parameters: None.

Returns: Nothing (procedure).

Notes: None.

whereClauseBuffer (combo.p)

Returns the buffername of a WHERE clause expression. This function avoids problems with leading or double blanks in WHERE clauses.

Parameters:

```
INPUT pcWhere AS CHARACTER
```

Complete where clause for ONE table with or without the FOR keyword. The buffer name must be the second token in the clause, not counting any FOR. For example: "EACH order OF Customer", but "FOR EACH order".

Returns: CHARACTER PRIVATE

Notes: PRIVATE, used internally in query.p only.

whereClauseBuffer (lookup.p)

Returns the buffer name of a WHERE clause expression. This function avoids problems with leading/double blanks in WHERE clauses.

Parameters:

```
INPUT pcWhere AS CHARACTER
```

The complete WHERE clause, with or without the FOR keyword, for a single table. The buffer name must be the second token (third if FOR is present) in the clause. For example: "EACH order OF customer", "FOR EACH order".

Returns: CHARACTER PRIVATE

Notes: PRIVATE, used internally in query.p only.

whereClauseBuffer (query.p)

Returns the buffer name of a WHERE clause expression. This function avoids problems with leading or double blanks in WHERE clauses.

Parameters:

```
INPUT pcWhere AS CHARACTER
```

The complete WHERE clause for **one** table, with or without the FOR keyword. The buffer name must be the second token in the where clause as in “EACH order OF Customer” or if “FOR” is specified the third token as in “FOR EACH order”.

Returns: CHARACTER PRIVATE

Notes: PRIVATE, used internally in `query.p` only.

windowDropDownList (toolbar.p)

Build or rebuild drop-down list of session windows. Loops through the existing list, deleting, then calls WindowListMenu and returns TRUE if there are any windows in the new list. Returns without action if in design mode.

Parameters:

```
INPUT pcBand AS CHARACTER
```

The submenu within which to build the list

Returns: LOGICAL

Notes: None.

windowEnabled (toolbar.p)

Checks whether some window is enabled (for window drop down list)

Parameters:

`INPUT phHandle AS HANDLE`

The window of interest

`OUTPUT plEnabled AS LOGICAL`

TRUE if it is enabled

Returns: Nothing (procedure).

Notes: None.

windowFocus (toolbar.p)

Applies ENTRY to the window, moves it to the front, and restores its usual size if minimized.

Parameters:

`INPUT phHandle AS HANDLE`

The window of interest

Returns: Nothing (procedure).

Notes:

windowListMenu (toolbar.p)

Builds a drop-down menu that lists session windows

Parameters:

`INPUT pcBand AS CHARACTER`

The submenu within which to build the list

`INPUT phContainer AS HANDLE`

The handle of the toolbar container

`INPUT phStart AS HANDLE`

The first window to be included

`INPUT-OUTPUT piCount AS INTEGER`

On output, the number of windows in the list

Returns: Nothing (procedure).

Notes: Recursive procedure!

SmartObject API Reference For WebSpeed

The WebSpeed environment coexists with and makes use of HTML. In this respect it is different to the standard 4GL/ADM environment, and objects operating in it—including SmartObjects—require a special set of supporting routines. Those routines are documented here.

Paths To Source Files

The default paths to the files defining the SmartObjects for WebSpeed are:

- **Class SourcePath** — src/web2
- **Class RcodePath** — web2
- **Class TemplatePath** — src/web2/template

Alphabetical Listing Of WebSpeed-specific API Routines

This section lists all SmartObject API routines dedicated to the WebSpeed environment. For ADM routines not specific to this environment, see the "ADM2 SmartObject API Reference". For lists of routines organized by source file, see the Index.

addColumnLink (wbtable.p)

Specifies that a column in the HTML table is to appear with a hyperlink. The necessary attributes to make this happen are specified in the other input parameters.

Parameters:

`INPUT *pcColumn* AS CHARACTER`

The name of the column that will be hyperlinked.

`INPUT *pcURL AS CHARACTER*`

The linked object. Must be a valid URL reference.

`INPUT *pcTarget AS CHARACTER*`

The HTML frame reference that will receive the response. "_self" specifies that the current HTML frame should be used.

`INPUT *pcMouseOver* AS CHARACTER`

A function that returns a character string that will be displayed on the mouseOver event in the HTML page.

`INPUT *pcJoinParam AS CHARACTER*`

Specifies which parameters must be added to the URL in order to join this data-source to the linked object's data-source. Valid values are:

- **"ROWID"** — Adds the parameters "ExternalRowids" with the current rowids as data to the URL, as well as "ExternalTables" with the Table property as data.
- **blank** — No join needs to take place and no parameters except the default "BackRowids" are added to the URL. Usually used to return to the calling object.

Returns: LOGICAL

Notes:

- The Wizard creates a link for one selected column, but the function is able to add links to all the columns in the object.
- The Embedded SpeedScript (.htm) files created by the Report wizard show examples of this function call in use.

- The actual HTML code to generate the link is in urlLink with the logic that adds join parameters in urlJoinParams. If the linked object (pcURL) is unspecified, the link will not be generated in the HTML code.
- The parameters ExternalTables and BackRowids are always added to the specified URL, while ExternalObject is added when the data-source is a SmartDataObject.
- The Wizard creates a link for one selected column, but the function is able to add links to all the columns in the object.
- The Embedded SpeedScript (.htm) files created by the Report wizard show examples of this function call in use.
- The actual HTML code to generate the link is in urlLink with the logic that adds join parameters in urlJoinParams. If the linked object (pcURL) is unspecified, the link will not be generated in the HTML code.
- The parameters ExternalTables and BackRowids are always added to the specified URL, while ExternalObject is added when the data-source is a SmartDataObject.
- The linked object must be able to interpret the data specified in the pcJoinParams. This is usually done by specifying that the linked object should be able to use this object's data-source as an "External table or object" when the object is created in the Wizard. This will generate a comma-separated ExternalTableList property in the linked object, with either a matching pipe-separated ForeignFieldList or two ExternalWhereList and ExternalJoinList properties. The ForeignFieldList property must have an entry that defines a ForeignFields property for the columns that are passed on this link, while the two others have matching entries that specify how the external tables are joined to the query.
- The list of columns is the only way to link objects that have SmartDataObjects as data-sources, but can also be used for database queries.
- When the "ROWID" option is used, the external tables are physically added to the receiving dynamic query if they are not already present, and the query–prepare is changed to use the corresponding rowids in the WHERE clause for these tables.
- The pcMouseOver needs to reference a function so that the mouse-over text can show current values for the actual row. This can be achieved by letting the function call columnProps in the data-source. The wizard will generate this function to return the text that is specified in the "Status" field in the wizard. You can refer to functions or properties in the actual data-source in this field inside a pair of back-ticks. The code generator will generate this into a valid run-time expression.

- All the URL link parameters, including the list of columns, will be added to the linked object's contextFields property. This guarantees that the linked object stays joined on subsequent requests to perform navigation, transaction, or search.

- All the parameters passed to this function will be added to internal properties that will be used when the HTML code is generated. The column name will be added to the LinkColumns, which is a comma-separated property that holds the name of all the linked columns. The other parameter passed to this function is added to other internal properties the LinkURLs, LinkTargets, LinkTexts, and LinkJoins respectively. All of them are comma-separated except the last one, which is CHR(1)-separated.

Examples:

```
/* Add two hyperlinks, one on the custNum to call the customer detail */
/* and one on the salesRep field to call the salesrep detail */
/* The custStatusLine and salesStatusLine is defined functions */

addColumnLink('custNum',         /* Column name */
              'detcust.html',    /* WebObject   */
              '_self',           /* TargetFrame */
              'custStatusLine',  /* Function name for Status Line */
              'custNum').        /* column to use as foreign field */

addColumnLink('salesRep',         /* Column name */
              'detsls.html',     /* WebObject   */
              '_self',            /* TargetFrame */
              'salesStatusLine',  /* Function name for Status Line */
              'salesRep').        /* column to use as foreign field */

FUNCTION custStatusLine RETURNS CHARACTER ():
  RETURN "Show detail for customer " + columnStringValue("custNum") + ".".
END.

FUNCTION salesStatusLine RETURNS CHARACTER ():
  RETURN "Show detail for salesrep " + columnStringValue("salesRep")
     + ".".
END.
```

addContextFields (webrep.p)

Adds fields to the list of fields that are used to keep context for the next request.

Parameters:

```
INPUT pcNewContextFields AS CHARACTER
```

List of new URL parameters to add to the list.

Returns: LOGICAL

Notes:

- The Property should be used whenever the HTML page needs to store the context.
- The Embedded SpeedScript templates already do this on all the URLs that are generated.

addSearchCriteria (webrep.p)

Adds SearchName and SearchValue to the data-source query.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The Column’s name in the data-source.

```
INPUT pcValue AS CHARACTER
```

Search value.

Returns: LOGICAL

Notes: None

addTDModifier (wbtable.p)

Adds one or more HTML attributes for the <TD> tag of a specific column.

Parameters:

`INPUT *pcColumn* AS CHARACTER`

The name of the column that will use the attribute.

`INPUT *pcModifier AS CHARACTER*`

One or more HTML attributes where the value is in double quoted HTML format.

Returns: LOGICAL

Notes:

- The pcModifier passed to this function will be added to any existing TD modifier attributes for this column. The actual data is stored in the internal TdModifier property. This is a comma-separated list that holds entries for each of the columns in the object.
- The HTML attributes must be in the format <attribute> = "<value>", which makes it necessary to use either single quotes around the character expression or a tilde before the double quote.

Examples:

```
/* Specify a HTML background color for the custNum column */
addTDModifier("custnum",' align="left" bgcolor = "#FFFa00"').
```

anyMessage (webrep.p)

Determines whether there are any messages in the message queue.

Parameters: None

Returns: LOGICAL

Notes:

- If it returns TRUE something has generated an error and a corresponding message.
- If there is a handle for the data-source, check **both** there and internally, because WebSpeed-specific errors are always stored internally.

assignColumnFormat (webrep.p)

Sets a column’s format.

Parameters:

`INPUT pcColumn AS CHARACTER`

The column’s name in the data-source.

`INPUT pcFormat AS CHARACTER`

The new format.

Returns: LOGICAL

Notes: Currently this is **not** a local override when a SmartDataObject is used, unless the SmartDataObject is destroyed on each request.

assignColumnHelp (webrep.p)

Overrides default HELP.

Parameters:

`INPUT pcColumn AS CHARACTER`

The column’s name in the data-source.

`INPUT pcHelp AS CHARACTER`

The new help text.

Returns: LOGICAL

Notes: Currently this is **not** a local override when a SmartDataObject is used, unless the SmartDataObject is destroyed on each request.

assignColumnLabel (webrep.p)

Overrides the default label.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column’s name in the data-source.

```
INPUT pcLabel AS CHARACTER
```

The new label.

Returns: LOGICAL

Notes: Currently this is **not** a local override when a SmartDataObject is used, unless the SmartDataObject is destroyed on each request.

assignColumnWidth (webrep.p)

Unused function.

Parameters:

```
INPUT pColum AS CHARACTER
```

```
INPUT pWidth AS DECIMAL
```

Returns: LOGICAL

Notes: Currently not in use.

assignExtentAttribute (webrep.p)

Enables the user to override field attributes for columns with extents.

Parameters:

`INPUT pcHandle AS HANDLE`

Buffer–field handle.

`INPUT piExtent AS INTEGER`

Extent.

`INPUT pcList AS CHARACTER`

The columns attribute, used to store ALL entries if there are differences.

`INPUT pcValue AS CHARACTER`

The new value.

Returns: CHARACTER PRIVATE

Notes:

- The buffer handle stores one value for help, label, and width. This is used to be able to have different attributes for each extent.
- Currently used for HELP and LABEL.
- The function is currently private. This must change if needed for valexp in `wbdata.p`.

assignFields (html-map.p)

Assigns the current form buffer values in the enabled fields/objects to the SmartDataObject, the database, and/or to local object variables.

Parameters: None

Returns: Nothing (procedure)

Examples:

```
PROCEDURE process-web-request:
  IF REQUEST_METHOD = "POST":U THEN DO:
     /* Copy HTML input field values to the form buffer fields. */
     RUN inputFields.
     /* Find the datasource record that needs to be assigned. */
     RUN findRecords.
     /* Assign form buffer field values to the datasource. */
     RUN assignFields.
     /* Display datasource field values to the form buffer fields. */
     RUN displayFields.
     /* Enable form buffer fields. */
     RUN enableFields.
     /* Output the static HTML page and form buffer field values to the web
stream. */
     RUN outputFields.
  END.
END.
```

Notes: Commonly used in a HTML Mapping object's process-web-request procedure.

assignFields (wbdata.p)

Retrieves the object's column values from the Web and saves them in the data-source. The AddMode property determines whether the data will be saved to the current record or to a new one.

Parameters: None

Returns: Nothing (procedure)

Notes:

- The DataColumns property defines the contents of the webObject and determines which columns to assign.
- The get–field with the
- name property as input parameter is used to retrieve the data value.
- This procedure is normally not called directly, but by ProcessWebRequest, which uses the values of the two Web fields "MaintOption" and "AddMode" to decide its actions. The web page usually has a set of transaction buttons that all share the name "MaintOption". This procedure will be called when this value is "submit". The web page will also have a hidden field, "AddMode," that serves as a context keeper. This field would have been set to "YES" by the previous request, if the object now is in add mode.
- If the data-source is a SmartDataObject, all columns (unless the columnReadOnly property returns TRUE) will be passed along with their corresponding values using the SmartDataObject's submitRow function.
- If the data-source is a database, this procedure starts the actual transaction when validateColumns() returns TRUE. The values retrieved from the Web will be assigned directly to the buffers by assigning the BUFFER–VALUE attribute.
- To create a new record, the AddMode property must be set to TRUE before calling this routine. When the data-source is a SmartDataObject, the SmartDataObject will be set in add mode by a call to the SmartDataObject's addRow function. The returned RowObject ROWID will be used as the first parameter to the submitRow(). When the data-source is a database, the LockRow() function that is called within the transaction block to upgrade the lock, will create a new record when the AddMode is TRUE.

assignTDModifier (wbtable.p)

Assigns one or more HTML attributes for the <TD> tag of a specific column.

Parameters:

`INPUT *pcColumn* AS CHARACTER`

The name of the column that will use the attribute.

`INPUT *pcModifier AS CHARACTER*`

One or more HTML attributes where the value is in double-quoted HTML format.

Returns: LOGICAL

Notes:

- The pcModifier passed to this function will overwrite any existing TD modifier attributes for this column. The actual data is stored in the internal TdModifier property. This is a comma-separated list that holds entries for each of the columns in the object.
- The HTML attributes must be in the format <attribute> = “<value>”, which makes it necessary to use either single quotes around the character expression or a tilde before the double quote.

Examples:

```
/* Specify a HTML background color for the custNum column */
assignTDModifier("custnum",'align="left" bgcolor = "#FFFa00"').
```

bufferHandle (html-map.p)

Gets the handle of a buffer by name.

Parameters:

`INPUT pcTableName AS CHARACTER`

The name of a table in the query or object.

Returns: HANDLE

Notes: None

bufferHandle (webrep.p)

Gets the handle of a buffer by table name.

Parameters:

```
INPUT pcTableName AS CHARACTER
```

The name of a table in the query.

Returns: HANDLE

Notes: None

columnDataType (webrep.p)

Returns a column’s datatype from the data-source.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The Column’s name in the data-source.

Returns: CHARACTER

Notes: None

columnFormat (webrep.p)

Returns a columns format from the data-source.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column’s name in the data-source.

Returns: CHARACTER

Notes: None

columnHandle (webrep.p)

Returns the buffer handle of a column in the internal query.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column’s name in the data-source.

Returns: HANDLE

Notes: This function is not used if the DataSource is a SmartDataObject.

columnHelp (webrep.p)

Return a column’s help text from the data-source.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column’s name in the data-source.

Returns: CHARACTER

Notes: None

columnHTMLName (html-map.p)

Finds the HTML field name of a column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Column name

Returns: CHARACTER

Notes: None

Examples:

```
/* Loop through datasource columns. */
DO iFld = 1 TO NUM-ENTRIES(cColumns):
    ASSIGN
       cColName   = ENTRY(iFld,cColumns).
     IF DYNAMIC-FUNCTION('columnReadOnly':U IN hDataSource, cColName)
     THEN NEXT.
     /* Build CHR(1)-separated list of column and HTML names. */
     ASSIGN
      cHTMLName  = DYNAMIC-FUNCTION('columnHTMLName':U
                      IN TARGET-PROCEDURE,cColName)
      cColString = cColString
                     + (IF cColString = "":U THEN "":U ELSE CHR(1))
                     + cColName + CHR(1) + get-field(cHTMLName).

END.
```

columnHTMLName (webrep.p)

Returns a unique valid HTML identifier/name for the column.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column's name in the data-source.

Returns: CHARACTER

Notes: Replaces "." and "-" in field name so it can be used as valid HTML objects. JavaScript cannot operate on objects with dashes or periods in the name.

columnLabel (webrep.p)

Returns the Column's label.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column's name in the data-source.

Returns: CHARACTER

Notes: None

columnReadOnly (webrep.p)

Returns TRUE if the column is non updatable.

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column's name in the data-source.

Returns: LOGICAL

Notes: None

columnStringValue (webrep.p)

Returns the character value of the column named in the input parameter, or the Initial Value from the data dictionary if the UpdateMode flag is set to "ADD."

Parameters:

```
INPUT pcColumn AS CHARACTER
```

The column's name in the data-source.

Returns: CHARACTER

Notes: The UpdateMode flag is set with the following call:

```
setUpdateMode('ADD').
```

For additional information, see the UpdateMode entry in the"Properties Glossary."

columnTable (html-map.p)

Overrides to handle HTML mapping objects with neither query nor SDO

Parameters:

```
INPUT pcColumn AS CHARACTER
```

Returns: CHARACTER

Notes: None

columnTDModifier (wbtable.p)

Returns the HTML attributes for the <TD> tag of a specific column.

Parameters:

`INPUT pcColumn AS CHARACTER`

The name of a column.

Returns: CHARACTER

Notes:

- This function is intended for internal use when the <TD> tags for the HTML columns are generated.
- The actual data is stored in the internal TdModifier property. This is a comma-separated list that holds entries for each of the columns in the object.

columnValMsg (wbdata.p)

Returns the validation message of a particular column. This is the text that serves as the error message when a validation fails.

Parameters:

`INPUT pcColumn`

The name of a column in the data-source. Can be qualified with table name and/or database name when the data-source is a database.

Returns: CHARACTER

Notes:

- The function is data-source transparent and calls the function with the same name in the data-source when the data-source is a SmartDataObject.
- The value returned by this function will be added to the error message when validateColumns encounters a column that fails validation.

constructObject (html-map.p)

Runs from adm–create–objects to run a SmartObject and to establish its parent and initial property settings. This is used internally to start the SmartDataObject that is the dataSource and set the instance properties.

Parameters:

`INPUT pcProcName AS CHARACTER`

Procedure name to run.

`INPUT phParent AS HANDLE`

Not used. INPUT pcPropList AS CHARACTER

Property list to set.

`OUTPUT phObject AS HANDLE`

New procedure handle.

Returns: Nothing (procedure)

Notes: Related routine: adm–create–objects

dataAvailable (html-map.p)

Overrides dataAvailable

Parameters:

`INPUT pcMode AS CHARACTER`

See `query.p`

Returns: Nothing (procedure)

Notes: This is a workaround to avoid running `query.p` logic when this object uses a data-source.

deleteBuffer (wbdata.p)

Deletes the current record of a query's buffer from the database. Returns TRUE when it succeeds and FALSE if it fails.

Parameters:

```
INPUT Buffer AS HANDLE
```

The current record's buffer handle.

Returns: LOGICAL

Notes:

- This function is only used when the data-source is a database.
- It is called from deleteRow, inside of the transaction block, for each of the query buffers returned by the DeleteTables property.
- The default function uses the BUFFER–DELETE method on the passed handle.
- The main purpose of deleteBuffer is to override BUFFER–DELETE for the deletion of a particular table. This is necessary when a table has a schema validation. The BUFFER–DELETE that usually deletes dynamic buffer records does not delete these tables. The AppBuilder detail wizard will generate the necessary override function in the target–procedure's source.
- A message will be added to the ADM 2 message queue if the delete fails.

deleteOffsets (html-map.p)

Deletes all HTML field offset temp-table records for a HTML mapping procedure.

Parameters: None

Returns: Nothing (procedure)

Notes: Intended for internal use only.

deleteRow (wbdata.p)

Delete the current row from the data-source.

Parameters: None

Returns: LOGICAL

Notes:

- The function is data-source transparent and calls the function with the same name when the data-source is a SmartDataObject.
- If the data-source is a SmartDataObject, it will get its RowObject ROWID and use it as an input parameter to the SmartDataObject's deleteRow.
- If the data-source is a database, it uses the table names returned from the DeleteTables property and passes the buffer's handle found in the query to the deleteBuffer function.

destroy (admweb.p)

Deletes this Web object's associated offset records in memory and this object if it was run PERSISTENT.

Parameters: None

Returns: Nothing (procedure)

Notes: None

destroyDataObject (webrep.p)

Shuts down the current data object.

Parameters: None

Returns: LOGICAL

Notes: None

destroyObject (admweb.p)

Destroys the procedure including the offset.

Parameters: None

Returns: Nothing (procedure)

Notes: None

destroyObject (webrep.p)

Performs any necessary cleanup before the object is destroyed.

Parameters: None

Returns: Nothing (procedure)

Notes: The main purpose of this procedure is not to delete the TARGET–PROCEDURE, but to perform specified cleanup when the data-source is a SmartDataObject. The SmartDataObject Instance Properties DestroyStateless and DisconnectAppserver determine whether the SmartDataObject is to be destroyed or disconnected.

disconnectObject (webrep.p)

Disconnects from the AppServer if there is one.

Parameters: None

Returns: Nothing (procedure)

Notes: First disconnectObject does an explicit destroyObject on the AppServer to give that object an opportunity to clean up. This procedure is invoked from destroyObject or can be run directly to disconnect without exiting.

dispatchUtilityProc (html-map.p)

Calls the standard utility procedure, as defined in `tagmap.dat`, for the current Web object.

Parameters:

`INPUT p_method AS CHARACTER`

The method procedure to run in the utility procedure (WEB.INPUT, WEB.OUTPUT).

`INPUT p_field-hdl AS HANDLE`

Progress object handle.

`INPUT p_field-data AS CHARACTER`

Data to send to the procedure.

`INPUT p_item-counter AS INTEGER`

Radio-set item to process.

`OUTPUT p_result AS LOGICAL`

Indicates method ran successfully

Returns: Nothing (procedure)

Notes: Intended for internal use only.

displayFields (html-map.p)

Copies SmartDataObject or database values to the displayed object and field form buffer screen values.

Parameters: None

Returns: Nothing (procedure)

Notes: Commonly used in process-web-request procedure.

Examples:

```
PROCEDURE process-web-request:
  IF REQUEST_METHOD = "POST":U THEN DO:
    /* Copy HTML input field values to the form buffer fields. */
    RUN inputFields.
    /* Find the datasource record that needs to be assigned. */
    RUN findRecords.
    /* Assign form buffer field values to the datasource. */
    RUN assignFields.
    /* Display datasource field values to the form buffer fields. */
    RUN displayFields.
    /* Enable form buffer fields. */
    RUN enableFields.
    /* Output the static HTML page and form buffer field values to the
    web stream. */
    RUN outputFields.
  END.
END.
```

enableFields (html-map.p)

Enables object and field form buffer widgets by setting their SENSITIVE attribute to TRUE.

Parameters: None

Returns: Nothing (procedure)

Notes: Commonly used in process–web–request procedure.

Examples:

```
PROCEDURE process-web-request:
   IF REQUEST_METHOD = "POST":U THEN DO:
    /* Copy HTML input field values to the form buffer fields. */
    RUN inputFields.
    /* Find the datasource record that needs to be assigned. */
    RUN findRecords.
    /* Assign form buffer field values to the datasource. */
    RUN assignFields.
    /* Display datasource field values to the form buffer fields. */
    RUN displayFields.
    /* Enable form buffer fields. */
    RUN enableFields.
    /* Output the static HTML page and form buffer field values to the
web stream. */
    RUN outputFields.
  END.
END.
```

exclusiveLockBuffer (wbdata.p)

Exclusively locks a buffer in the query.

Parameters:

```
INPUT phBuffer AS HANDLE
```

Buffer being locked.

Returns: LOGICAL PRIVATE

Notes: Defined as private because its only used by deleteBuffer and lockRow, and there is no check whether this is a table in the query.

extentAttribute (webrep.p)

Returns the attribute for a variable with extent.

Parameters:

`INPUT piExtent AS INTEGER`

Extent.

`INPUT pcList AS CHARACTER`

The columns attribute, which is used to store ALL entries if any one is different.

Returns: CHARACTER PRIVATE

Notes:

- The buffer handle stores one value for help, label, and so forth. To have different attributes for each extent, they are stored with delimiter CHR(3).
- Used for LABEL and HELP.
- Currently private. This must change if it is needed for ValExp.

extentValue (webrep.p)

Returns the extent number if a field name has brackets.

Parameters:

`INPUT pcColumn AS CHARACTER`

The column’s name in the data-source.

Returns: INTEGER

Notes: Returns 0 if no brackets in the name.

fetchCurrent (webrep.p)

Repositions the data-source query to the rowids stored in the CurrentRowids property.

Parameters: None

Returns: Nothing (procedure)

Notes:

- This procedure is called from fetchNext and fetchPrev in order to re-establish context before the actual navigation takes place.
- The Property CurrentRowids is immediately set from the Web field with the same name to keep the context between each request.

fetchFirst (webrep.p)

Repositions the data-source query to the first row.

Parameters: None

Returns: Nothing (procedure)

Notes: None

fetchLast (wbtable.p)

Repositions the query of the data-source to the first row of the last page.

Parameters: None

Returns: Nothing (procedure)

Notes:

- This procedure is an override of fetchLast in `webrep.p` and contains the logic necessary to move the cursor backward to the row that should appear on the top of the last page. It does this by calling the PageBackward function after it has executed SUPER to reposition to what is the true last row of the data-source.
- The reason that this function is named fetchLast and not fetchLastPage to simplify the interface and conceal navigation details from the caller.

fetchLast (webrep.p)

Repositions the data-source to the query's last row.

Parameters: None

Returns: Nothing (procedure)

Notes: None

fetchNext (wbtable.p)

Repositions the query of the data-source to the first row of the next page.

Parameters: None

Returns: Nothing (procedure)

Notes:

- This procedure totally overrides the fetchNext in `webrep.p` and contains the logic necessary to move the cursor forward from the Web page's current row to reposition to the row that appears on the top of the next page.
- The reason that this function is named fetchNext and not fetchNextPage is to simplify the interface and conceal navigation details from the caller.

fetchNext (webrep.p)

Repositions the data-source query to the next row.

Parameters: None

Returns: Nothing (procedure)

Notes: The procedure re-establishes context with a call to fetchCurrent before the actual navigation takes place.

fetchPrev (wbtable.p)

Repositions the query of the data-source to the first row of the previous page.

Parameters: None

Returns: Nothing (procedure)

Notes: This procedure is an override of fetchPrev in `webrep.p` and contains the logic necessary to move the cursor backward to the row that should appear on the top of the previous page. It does this by calling the PageBackward function after it has executed SUPER to find the current row received from the Web page, and repositions to what is the current row's true previous row. The reason that this function is named fetchPrev and not fetchPrevPage is to simplify the interface and conceal navigation details from the caller.

fetchPrev (webrep.p)

Repositions the data-source query to the previous row.

Parameters: None

Returns: Nothing (procedure)

Notes: The procedure re-establishes context with a call to fetchCurrent before the actual navigation takes place.

fieldExpression (webrep.p)

Creates an expression for a field.

Parameters:

```
INPUT pColumn AS CHARACTER

INPUT pOperator AS CHARACTER

INPUT pValue AS CHARACTER
```

Returns: CHARACTER

Notes: None

findRecords (html-map.p)

Opens the SmartDataObject or database query that finds the records in this frame.

Parameters: None

Returns: Nothing (procedure)

Notes: Commonly used in process–web–request procedure.

Examples:

```
PROCEDURE process-web-request:
   IF REQUEST_METHOD = "POST":U THEN DO:
    /* Copy HTML input field values to the form buffer fields. */
    RUN inputFields.
    /* Find the datasource record that needs to be assigned. */
    RUN findRecords.
    /* Assign form buffer field values to the datasource. */
    RUN assignFields.
    /* Display datasource field values to the form buffer fields. */
    RUN displayFields.
    /* Enable form buffer fields. */
    RUN enableFields.
    /* Output the static HTML page and form buffer field values to the
       web stream. */
    RUN outputFields.
  END.
END.
```

htmAssociate (html-map.p)

Maps HTML fields to their Web object widget counterparts.

Parameters:

`INPUT htmField AS CHARACTER`

HTML field name.

`INPUT wdtField AS CHARACTER`

Web object field name.

`INPUT widHandle AS HANDLE`

Web object field handle.

Returns: Nothing (procedure)

Notes: Internal use only. AppBuilder-maintained, read-only. The essential HTML ”mapping” procedure.

HTMLAlert (wbrep.p)

Generates an alert-box in the HTML.

Parameters:

`INPUT pcMessage AS CHARACTER`

The message to display.

Returns: LOGICAL

Notes: The message is generated with JavaScript.

HTMLColumn (wbtable.p)

Outputs the columns value within an HTML <TD> tag, including the HREF definitions to create defined hyperlinks.

Parameters:

`INPUT pcColumn`

The name of the column to output.

Returns: LOGICAL

Notes: Checks the LinkColumns property to see if any link attributes are defined. An HREF will be generated to create the hyperlink if an object is specified. If a function is defined for the status line, it will be called and the returned text will be generated in a mouse-over event. The urlLink function is called to add necessary link parameters to the HREF link.

HTMLSetFocus (wbrep.p)

Sets focus to a field in a WebPage.

Parameters:

```
INPUT pcForm AS CHARACTER
```

The NAME of the HTML form of the field.

```
INPUT pcColumn AS CHARACTER
```

Progress column name.

Returns: LOGICAL

Notes:

- The generated code is JavaScript.
- The columnHTMLName function returns a valid HTML field name for the passed Column.

HTMLTable (wbtable.p)

Outputs the entire HTML <TABLE>.

Parameters: None

Returns: LOGICAL

Notes:

- Checks the datatype of the column and adds the HTML attribute to right align decimals and integers with the help of the addTDModifier function
- Adds labels to the HTML table if UseColumnLabels is set to TRUE.
- Calls the HTMLColumn function for each column. This is done as many times as specified in the TableRows property.

initializeObject (html-map.p)

Sets the list of displayed and enabled objects/fields.

Parameters: None

Returns: Nothing (procedure)

Notes: Intended for internal use only.

inputFields (html-map.p)

Receives field input from the Web browser and populates the form buffer field values.

Parameters: None

Returns: Nothing (procedure)

Notes: Runs the WEB.INPUT procedure for each HTML field in the offset (`.off`) file. Commonly used in process–web–request procedures.

Examples:

```
PROCEDURE process-web-request:
   IF REQUEST_METHOD = "POST":U THEN DO:
    /* Copy HTML input field values to the form buffer fields. */
    RUN inputFields.
    /* Find the datasource record that needs to be assigned. */
    RUN findRecords.
    /* Assign form buffer field values to the datasource. */
    RUN assignFields.
    /* Display datasource field values to the form buffer fields. */
    RUN displayFields.
    /* Enable form buffer fields. */
    RUN enableFields.
    /* Output the static HTML page and form buffer field values to the
    web stream. */
    RUN outputFields.
  END.
END.
```

joinExternalTables (wbrep.p)

Adds external tables to the buffers in the database query.

Parameters:

`INPUT pcTables AS CHARACTER`

A comma-separated list of tables to join.

`INPUT pcRowids AS CHARACTER`

The corresponding list of ROWIDS.

Returns: LOGICAL

Notes: There are two ways to treat a list of external tables where one or more of the tables already are in the query:

- As a default use the corresponding ExternalRowid in the where clause in the query. All other tables are disregarded.
- If the external table list is defined in the ExternalTables attribute, disregard the tables that are in the query and join to the OTHER tables. (ExternalJoinList and ExternalWhereList already have the right number of entries.)

joinForeignFields (wbrep.p)

Finds the foreign fields to use for the passed external tables or object and creates the new query statement in the data-source. Gets the values from the Web and adds the columns and values to the query.

Parameters:

```
INPUT pcObject AS CHARACTER
```

A table or SmartDataObject name that has an entry in the ForeignFieldsList.

Returns: LOGICAL

Notes: None

lockRow (wbdata.p)

Locks all the records for the query or creates records for the tables in the query if the AddMode property is set to TRUE. Returns FALSE if it fails to lock the record for some reason.

Parameters:

```
INPUT pLock AS CHARACTER
```

If the option begins “EXCLUSIVE” the records will be locked exclusively. Otherwise the records will be NO–LOCKed.

Returns: LOGICAL

Notes:

- The function is only used for database data-sources.
- The function will return FALSE and add the necessary error messages to the ADM 2 message queue if the record is unavailable or locked.
- Used internally by assignFields and deleteRow and not intended for external use.
- Also used internally to reset the lock to NO–LOCK when a transaction ends.

openQuery (webrep.p)

Opens the database query in the data-source.

Parameters: None

Returns: LOGICAL

Notes: None

outputFields (html-map.p)

Replaces the tagged HTML field definition with the data values stored in the form buffer.

Parameters: None

Returns: Nothing (procedure)

Notes: Merges the HTML file with the results of running the WEB.OUTPUT procedure, if one exists, for each HTML field. Otherwise just the HTML field definition is output. When available, default utility procedures containing the WEB.OUTPUT procedure are run, based on HTML field type. Commonly used in process–web–request procedure.

Examples:

```
PROCEDURE process-web-request:
   IF REQUEST_METHOD = "POST":U THEN DO:
    /* Copy HTML input field values to form buffer fields. */
    RUN inputFields.
    /* Find datasource record that needs to be assigned. */
    RUN findRecords.
    /* Assign form buffer field values to datasource. */
    RUN assignFields.
    /* Display datasource field values to form buffer fields. */
    RUN displayFields.
    /* Enable form buffer fields. */
    RUN enableFields.
    /* Output static HTML page and form buffer field values to web stream. */
    RUN outputFields.
  END.
END.
```

pageBackward (wbtable.p)

Reads the previous record from the data-source the number of times required to find the row that should be the top row in the table in order to make the current record the last record.

Parameters: None

Returns: LOGICAL

Notes:

- This is all additional logic that makes fetchPrev and fetchLast override their SUPER.
- The function is called from fetchNext if the last row is encountered as fetchNext loops through the number of records specified in the TableRows property. It finds the row that should appear on the top of the next page.

processWebRequest (wbdata.p)

Processes the request from the Web. This is an override of the `webrep.p` version and has the additional logic necessary to handle transaction requests.

Parameters: None

Returns: Nothing (procedure)

Notes:

- The logic in this procedure is based on the values of the two Web fields “MaintOption” and “AddMode” which are maintained on the HTML page and received as part of the web request.
- The value of the Web field AddMode is passed to the object’s AddMode property to distinguish between the creation of a new record or a save of the current.
- When ”MaintOption” is “submit,” the assignFields procedure is called. When ”MaintOption” is “delete,” the deleteRow function is called.
- When “MaintOption” is “submit” or “delete” the value returned from the Rowids property is set to the CurrentRowids property **after** the request has been executed. This property is maintained in the HTML page as a context keeper and is usually maintained by the SUPER procedure. It needs to be updated here in case the current record was deleted or a new record has become the current.

- The procedure has logic to deal with the case of an empty query. It changes the object's mode to "add" by assigning the UpdateMode property's value to "add". This change is accompanied by an addMessage call that results in a corresponding message being generated in the HTML page.

processWebRequest (wbtable.p)

Processes the request from the Web. This is an override of the `webrep.p` version and contains additional logic to be able to produce an HTML page with several rows in a table.

Parameters: None

Returns: Nothing (procedure)

Notes:

- The actual HTML page is generated in the HTMLTable function
- Because the object is read-only it does not go into add mode when the query is empty. IN this case, an error message is added to the ADM 2 message queue.
- The procedure ends with a destroyObject call, which is important when the data-source is a SmartDataObject, even if the TARGET–PROCEDURE is not run persistently. The reason is that destroyObject might destroy the SmartDataObject and/or disconnect any AppServer connection depending on the Instance Properties of the SmartDataObject.

processWebRequest (webrep.p)

Processes the submitted request from the Web.

Parameters: None

Returns: Nothing (procedure)

Notes:

- The following Web data are retrieved with get–field:
  - **CurrentRowids** — A list of the rowids that are currently in use for this object on the Web
  - **Navigate** — Next,Prev,First,Last,Search.
  - **Maintoption** — Add,Delete,Submit.
  - **SearchValue**—Use when Navigate = "Search".
  - **ExternalObjects** — Data-source objects to join to (comma separated).
  - **ExternalTables** — Tables to join to (comma separated).
  - **ExternalRowids** — ROWIDs that correspond to ExternalTables ROWIDs (comma separated).
- After the data-source query has been manipulated according to the received Web data and other properties, the data-source query is opened.
- If this is a search and the query is empty, the query is reopened and a message is added to the ADM 2 message queue to inform the user that no data was found.
- The last action that is performed is to set the CurrentRowids property from the Rowids property. This must be done last because the property is used in navigation and because, to keep context for the next request, the property must reflect the new current record as it is going to be passed back to the Web page.

readOffsets (html-map.p)

Wrapper to a procedure that reads the HTML mapping offset file and populates internal AppBuilder temp-tables.

Parameters:

```
INPUT cWebFile AS CHARACTER
```

Name of offset file.

Returns: Nothing (procedure)

Notes: Internal use only.

removeEntry (webrep.p)

Removes an entry from a list.

Parameters:

```
INPUT pNum AS INTEGER
```

```
INPUT pList AS CHARACTER
```

Returns: CHARACTER PRIVATE

Notes:

- PRIVATE, not recommended for use.
- Delimiter is always comma.

reOpenQuery (webrep.p)

Resets the data-source query to its original state.

Parameters: None

Returns: LOGICAL

Notes: This is the state after context has been reset, but before a search or query has been performed.

rowidExpression (webrep.p)

Creates ROWID expression for dynamic query Rowid(table) = to–rowid(rowidchar).

Parameters:

`INPUT pcBuffer AS CHARACTER`

Buffer name in the query.

`INPUT pcRowid AS CHARACTER`

STRING of the Rowid.

Returns: CHARACTER PRIVATE

Notes: None

showDataMessages (webrep.p)

Does a fetchMessages to retrieve all data-related messages (normally database update-related error messages) and calls the HTMLAlert function to show them in an alert box on the Web.

Parameters: None

Returns: CHARACTER

Notes:

- Returns the name of the field (if any) from the first error message, to allow the caller to use it to position the cursor.
- This procedure expects to receive back a single string from fetchMessages with one or more messages delimited by CHR(3), and within each message the message text, field name (or blank) + a table name (or blank), delimited by CHR(4), if present.

startDataObject (webrep.p)

Starts or connects to the SmartDataObject. If the AppService attribute is set in this object, it must also be set in the SmartDataObject before that SmartDataObject is initialized.

Parameters:

```
INPUT pcDataSource AS CHARACTER
```

Procedure name of the SmartDataObject to be started (for example, `dcust.w`).

Returns: LOGICAL

Notes: The properties ”OpenOnInit”,”CheckLastOnOpen” and ”RebuildOnRepos” are always set to TRUE.

timingOut (admweb.p)

Sets a Web object’s Web state to timed–out.

Parameters: None

Returns: Nothing (procedure)

Notes: ”RUN timingOut” is equivalent to ”setWebState (0)”.

urlJoinParams (webrep.p)

Generate the URL parameter to use as join information for a linked object.

Parameters:

`INPUT pcJoinParam AS CHARACTER`

Specifies which parameters must be added to the URL in order to join this data-source to the linked object's data-source. Following are the valid values for this parameter:

- "ROWID" specifies that record information should be passed as rowids in the ExternalRowids URL parameter (only when using database as data-source).
- A comma-separated list of column names; for example, `"custnum,state,salesrep"`.
- Blank. No link information is needed.

Returns: CHARACTER

Notes:

- This is called from urlLink with the correct entry from the CHR(3)-delimited JoinLinks attribute.
- Add "?" as the last entry to the parameter to specify that the first parameter should be separated with "?" (the FIRST URL parameter).

urlLink (webrep.p)

Returns the necessary URL parameters in order to pass record information to a linked object.

Parameters:

`INPUT pcWebObject AS CHARACTER`

The object to call (can have URL parameters).

`INPUT pcJoinParam AS CHARACTER`

Specifies which parameters must be added to the URL in order to join this data-source to the linked object's data-source. Following are the valid values for this parameter:

- "ROWID" specifies that record information should be passed as rowids in the ExternalRowids URL parameter (only when using database as data-source).

- A comma-separated list of column names; for example, `"custnum,state,salesrep"`.
- Blank. No link information is needed.

Returns: CHARACTER

Notes: None

validateColumns (wbdata.p)

Returns TRUE if all of the database columns in the optionally defined frame are validated. The frame is generated specifically for this purpose.

Parameters: None

Returns: LOGICAL

Notes:

- The function is only used when the data-source is a database.
- The Embedded SpeedScript detail wizard optionally generates a frame containing all the database columns that have been specified to inherit data dictionary validation. This function does the actual parsing and validation of the database columns in that frame.
- The function retrieves the actual values to validate from the Web unless it finds fields in the EnabledFields Property. In that case, it assumes that the values already have been moved to the frame. The intention of this logic is to support HTML mapping objects.

  NOTE: The HTML mapping object does not call this default. The use of EnabledFields to indicate whether the values should be retrieved from the Web or already are in the frame is very likely to change.

validateColumnValue (webrep.p)

Checks if a value is the correct data type.

Parameters:

`INPUT pcColumn AS CHARACTER`

The column’s name in the data-source.

`INPUT pcValue AS CHARACTER`

Value to validate.

Returns: LOGICAL

Notes: Returns TRUE if Column Value is valid, FALSE if it is not.

Properties Glossary

A

ActionEvent

Returns the event to publish on Default-action of the browse. setActionEvent(pcEvent) defines the persistent trigger that runs defaultAction and also subscribes the source-procedure.

ActionGroups

Returns the action groups selected in the Instance Properties.

AddMode

Sets the ADDMode property. If set to TRUE, the assignFields procedure will create a new record.

AllFieldHandles

The pseudo-function {get} returns the allFieldHandles value.

AllFieldNames

The pseudo-function {get} returns the allFieldNames value.

ApplyActionOnExit

Uses the {get} pseudo-function to return the value of ApplyActionOnExit (true if exiting selects the browse row). Currently used by SmartSelect.

ApplyExitOnAction

Currently used by SmartSelect. The logic is not performed in the trigger, but in the defaultAction procedure that gets defined as a persistent DEFAULT-ACTION event when setActionEvent is defined. Local DEFAULT-ACTION events could be set up to run defaultAction.

AppService

1.) Returns the logical partition name of the AppService to be used to connect to an AppServer. This comes from a property, but there is no xp macro defined so as to avoid returning '(None)'.

2.) Returns the name of the stored AppService.

3.) Stores the AppService in which the SmartDataObject is to be started. This routine should be called before startDataObject. If the datasource is valid it must be disconnected it if it is connected to a different partition.

ASBound

Returns TRUE if this object binds the AppServer with a persistently running procedure, usually the server part of this object. The AsHandle property can check this.

AsDivision

The {get} pseudo-function returns the value of the ASDivision property. The value is a string indicating which side of the AppServer this Object is running on; client, server, or the empty string (no AppServer).

ASHandle

Returns the handle to this object's companion procedure (the copy of itself) running on the AppServer. The handle returned is the handle of the persistent procedure, not of the connection.

ASHasStarted

The {get} pseudo-function returns the value of ASHasStarted. This value is TRUE if the object has done its first call to its server-side object. There is no corresponding set function as this must only be set internally.

ASInfo

The {get} pseudo-function returns the value of ASInfo. That value is the string used as a parameter when this object connects to its partition for the first time. The string will be empty if no parameters were passed.

ASInitializeOnRun

The {get} pseudo-function returns the value of ASInitializeOnRun. That value is TRUE if runServerObject should call initializeServerObject

initializeServerObject is called on the client, but will usually have a call to the server to set/retrieve context.

AssignList

The {get} pseudo-function returns the value of AssignList, the list of updatable columns whose names have been modified in the SmartDataObject. This string takes the form: <RowObjectFieldName>,<DBFieldName>[,...][CHR(1)...] with a comma-separated list of pairs of fields for each db table, and CHR(1) between the lists of pairs.

ASUsePrompt

The {get} pseudo-function returns the value of ASUsePrompt. That value is TRUE if the supporting code should prompt for a Username and Password when connecting to its Application Partition.

AsynchronousSDO

The {get} pseudo-function returns the value of AsynchronousSDO.

Attribute

Returns the value of a standard Web-related attribute. The name of the attribute. Possible names are type, version, web–state, web–timeout, web–timeout–handler, web–time–remaining. In addition, other names can be used, provided that the special–get–attribute procedure to handle them exists in the target procedure. For example:

```
RUN getAttribute ("web-state").
cWebState = RETURN-VALUE.
```

AutoCommit

The {get} pseudo-function returns the value of AutoCommit. This value determines whether a Commit happens on every record update. This is FALSE by default, but if set to TRUE will do a Commit automatically after any change.

AutoSort

Returns whether the root node should be auto-sorted.

AvailMenuActions

The {get} pseudo-function returns the value of AvailMenuActions, the actions available in this menu.

AvailToolbarActions

The {get} pseudo-function returns the value of AvailToolbarActions, the list of actions available in the menu of this toolbar. The Instance Properties dialog shows these and AvailMenuActions. The actions selected will be saved as ActionGroups.

AvailToolbarBands

Returns a comma-separated list of the available toolbar bands for this toolbar master. Assembles the list by looping through the temp-table.

B

BaseQueryString

The {get} pseudo-function returns the value of BaseQueryString.

BlockDataAvailable

The {get} pseudo-function returns the value of BlockDataAvailable, true if DataAvailable messages from contained SDOs are to be ignored and not republished

BoxRectangle

The {get} pseudo-function returns the value of BoxRectangle. This is the handle to the rectangle, if any, around the buttons in the Panel used by resizeObject.

BoxRectangle2

The {get} pseudo-function returns the value of BoxRectangle2. Used for bottom rectangle on toolbars where ToolbarAutosize is TRUE.

BrowseFieldDataTypes

The {get} pseudo-function returns the value of BrowseFieldDataTypes.

BrowseFieldFormats

The {get} pseudo-function returns the value of BrowseFieldFormats.

BrowseFields

The {get} pseudo-function returns the value of BrowseFields, the list of fields to display in the browse when the viewAs property is set to "browse".

BrowseHandle

The {get} pseudo-function returns the value of BrowseHandle, the handle of the browse control.

BrowseTitle

The {get} pseudo-function returns the value of BrowseTitle, the title to display in the browse SmartWindowContainer when the ViewAs property is set to "browse".

BuildSequence

The {get} pseudo-function returns the value of BuildSequence. A scoped xp macro is defined and undefined specifically for this {get}.

Buffer

Used to create or assign properties for an action.

Buffers

Sets the Buffers property (see `webrprop.i`), a comma-separated list of database tables (not SmartDataObject) to be retrieved by this object.

ButtonCount

The {get} pseudo-function returns the value of ButtonCount, the number of buttons in a SmartPanel. Used by resizeObject.

Buttons

Used to set buttons according to the current state. It is called from various link targets when they publish some state. Depending on the caller, valid values can be: onlyRecord, disable-nav, first, last, initial-tableio, update, add-only, disable-commit, enable-commit, enable-filter. The NAVIGATION-TARGET publishes queryPosition, the TABLEIO-TARGET publishes updateState, and the COMMIT-TARGET publishes rowObjectState.

C

CalculatedColumns

The {get} pseudo-function to examine the values of DataColumnsByTable and Tables. Returns a comma-delimited list of the calculated columns for the SmartDataObject.

CalcWidth

The {get} pseudo-function returns the value of CalcWidth. That value is the logical value that determines whether the width of the browse is calculated to the exact amount needed for the fields it contains.

CallerObject

The {get} pseudo-function returns the value of CallerObject.

CallerProcedure

The {get} pseudo-function returns the value of CallerProcedure.

CallerWindow

The {get} pseudo-function returns the value of CallerWindow.

CancelBrowseOnExit

The {get} pseudo-function returns the value of CancelBrowseOnExit, TRUE if the value in the browse is NOT to be selected on Exit.

CascadeOnBrowse

The {get} pseudo-function returns the value of CascadeOnBrowse. This determines whether data will be retrieved from a dependent SDO if the parent SDO has more than one row in its current dataset; if TRUE (the default), data will be retrieved for the first row in the parent dataset, otherwise not.

ChangedEvent

The {get} pseudo-function returns the value of ChangedEvent, an optional event to publish on value-changed.

CheckCurrentChanged

The {get} pseudo-function returns the value of CheckCurrentChanged. This value determines whether the DataObject code should check whether the database rows being updated have been changed since read.

CheckLastOnOpen

The {get} pseudo-function returns the value of CheckLastOnOpen, the flag indicating whether a get-last should be performed on an open in order for fetchNext to detect that we are on the last row. This is necessary to make the QueryPosition attribute reliable. Note:

- The DataColumsByTable property that stores in different tables delimited by CHR(1) instead of a comma in order to identify which columns are from which table in the event of a join. For example, if the query is a join of customer and order and the Name field from customer and the OrderNum and OrderData field from Order are selected, then the property value will be equal to "Name<CHR(1)>OrderNum,OrderDate".
- This function replaces CHR(1) with "," returns just a comma-delimited list.

ChildDataKey

The {get} pseudo-function returns the value of ChildDataKey.

ClientID

The {get} pseudo-function returns the value of clientID.

ClientProxyHandle

Stores the character version of the client-side SDO handle. The string containing the client-side SDO procedure handle.

ClientRect

Returns the client rectangle for the Treeview

Col

Returns the column of the object.

ColumnFormat

The {get} pseudo-function returns the value of ColumnFormat, the browse column format overrides values.

ColumnLabels

The {get} pseudo-function returns the value of ColumnLabels, the browse column label overrides values.

Columns

Sets the DataColumns property (see `webprop.i`), a comma-separated list of database tables (not SmartDataObject) to be retrieved by this object. A comma-separated list of column names. The set function is named setColumns instead of setDataColumns for backwards compatibility.

ComboDelimiter

The {get} pseudo-function returns the value of ComboDelimiter.

ComboFlag

The {get} pseudo-function returns the value of ComboFlag.

ComboFlagValue

The {set} pseudo-function assigns a new value to ComboFlagValue.

ComboHandle

The {get} pseudo-function returns the value of ComboHandle.

ComboQuery

The getComboQuery routine is published from the viewer and is used to pass the query required by this combo back to the viewer for building. Once built, the query will be returned into the procedure displayCombo. Note:

- This is designed to facilitate all combo queries being built with a single AppServer hit. It is published from displayfields in the viewer.
- This routine is not run at all in add mode.

ComboSort

The {set} pseudo-function assigns a new value to ComboSort. That value is the Sort propery for the Combo

CommitSource

The {get} pseudo-function returns the value of CommitSource. Used for pass-through for regular containers, but also inherited by the SBO, which uses it generally. A global xp macro is not defined because the SBO needs logic in override (set AutoCommit). To allow {set}, the function defines a temporary, scoped xp macro.

CommitSourceEvents

The {get} pseudo-function returns the value of CommitSourceEvents. This value represents the list of events to be subscribed to in the Commit Panel or other Commit-Source.

- Commit is a pass-through link, but the SBO uses it for real.
- This property is here because the link and event properties are kept together.

CommitTarget

The {get} pseudo-function returns the value of CommitTarget, the list in character format of the handles of this object's Commit-Targets.

CommitTargetEvents

The {get} pseudo-function returns the value of CommitTargetEvents. This value is the list of events to be subscribed to in the Commit Panel or other Commit-Target.

ConsumerSchema

Returns the XML Schema filename.

ContainedDataColumns

The {get} pseudo-function returns the value of ContainedDataColumns, a delimited list of all the DataColumns of all the Data Objects in this SBO.

ContainedDataObjects

The {get} pseudo-function returns the value of ContainedDataObjects, a list of the handles of the Data Objects contained in this SBO.

ContainerHandle

The {get} pseudo-function returns the value of ContainerHandle, the widget handle of this object's Window or Frame container. Returns: HANDLE.

ContainerHidden

The {get} pseudo-function returns the value of ContainerHidden, TRUE if this object's parent container is hidden.

ContainerMode

The {get} pseudo-function returns the value of ContainerMode.

ContainerSource

The {get} pseudo-function returns the value of ContainerSource, the handle of this object's Container-Source, if any.

ContainerSourceEvents

The {get} pseudo-function returns the value of ContainerSourceEvents, a comma-separated list of the events to which this object wants to subscribe to in its ContainerSource.

ContainerTarget

The {get} pseudo-function returns the value of ContainerTarget. This value is the list of the handles of the object's contained objects.

ContainerTargetEvents

The {get} pseudo-function returns the value of ContainerTargetEvents. This value is the comma-separated list of the events this object wants to subscribe to in its ContainerTarget.

ContainerType

The {get} pseudo-function returns the value of ContainerType, the type of container this SmartObject is ("Window" or "Frame"), or the empty string if the object is not a container.

ContextandDestroy

Server-side procedure to run after new data has been requested by the client.

ContextAndInitialize

Resets context and initializes this server-side object. Called from a stateless client before a request.

ContextFields

Retrieves the ContextFields property (see `webrprop.i`), a comma-separated list of Fields for which URL parameters are needed to keep context for subsequent requests to this object. Note:

- The Property should be used whenever the HTML page needs to append the context information to the URL being sent out as URL parameters. This can be seen in a Web report generated using the Report wizard in the following line:

```
ASSIGN TmpUrl= url-format(?,getContextFields(),?).
```

- The Embedded SpeedScript templates for both Report and Detail Web objects use this function. ExternalTables, ExternalRowIds, and Columns that are used as Foreignfields are examples of context fields.

ContextForServer

Sets a new value for gcContextForServer.

CreateHandles

The {get} pseudo-function returns the value of CreateHandles. This value is a comma-separated list of the handles, in character form, of the fields in the data visualization object that should be enabled for an Add or Copy operation.

CtrlFrameHandle

Returns the handle of the Treeview's control frame.

CurrentDescValue

The {get} pseudo-function returns the value of CurrentDescValue.

CurrentKeyValue

The {get} pseudo-function returns the value of CurrentKeyValue.

CurrentMessage

The {get} pseudo-function returns the value of CurrentMessage.

CurrentMessageId

The {get} pseudo-function returns the value of CurrentMessageID, the ID from the previous sendMessage where ReplyRequired.

CurrentPage

The {get} pseudo-function returns the value of CurrentPage. This value is the current page number of the Container.

CurrentRowids

1.) Returns the String of comma-delimited rowids in the CurrentRowid property. (See `webrprop.i`.). Note:

- The rowids stored in this property come from the URL parameter list and are requested in fetchCurrent.
- Rowids for the external tables are also stored in the CurrentRowid property. The function getTableRowids will return the rowids for the query's original tables.

2.) Sets the String of comma-delimited rowids in the CurrentRowid property that will be used by fetchCurrent. (See `webrprop.i`.) A list of ROWIDS. Note:

- Typically this would be set from a URL parameter or hidden field.
- The purpose of this property is to store the context of the current record received from the Web once, so that it only needs to RUN fetchCurrent whenever it is needed.

CurrentRowModified

Returns TRUE if any values in the current RowObject row have been modified. If there is no current RowObject record, then getCurrentRowModified returns "?".

D

DataColumns

The {get} pseudo-function to examine the value of DataColumnsByTable. Returns a comma-delimited list of the columnNames for the SmartDataObject.

- The DataColumsByTable property that stores in different tables delimited by CHR(1) instead of a comma in order to identify which columns are from which table in the event of a join. For example, if the query is a join of customer and order and the Name field from customer and the OrderNum and OrderData field from Order are selected, then the property value will be equal to "Name<CHR(1)>OrderNum,OrderDate".
- This function replaces CHR(1) with "," returns just a comma-delimited list.

DataColumns

The {get} pseudo-function returns the value of DataColumns, a list of all the DataColumns of all the Data Objects in this SBO, each qualified by the SDO ObjectName.

DataFieldDefs

The {get} pseudo-function returns the value of DataFieldDefs, the name of the include file in which the field definitions for this SDO's RowObject table are stored.

DataHandle

The {get} pseudo-function returns the value of DataHandle, the handle to the temp-table query.

DataHandle

This SBO version of getDataHandle is run from a browser to get the query from the contained Data object. Returns the handle of the RowObject query.

DataIsFetched

The {get} pseudo-function returns the value of DataIsFetched. The SBO sets this to TRUE in the SDO when it has fetched data on the SDO's behalf in order to prevent the SDO from another server call to fetch the data it already has. This is checked in `query.p` dataAvailable and openQuery is skipped if its TRUE. It is immediately turned off after it is checked.

DataLinksEnabled

The {get} pseudo-function returns the value of DataLinksEnabled.

DataLogicObject

Returns the global handle ghDataLogicProcedure.

DataModified

The {get} pseudo-function returns the value of DataModified. This value is TRUE if the current RowObject record is modified, or NO if there is no current RowObject.

- The difference from getNewRow is that it also returns true for saved and uncommitted new record and thus cannot be used to check the object's state. This uses the RowMod field in the temp-table to see if the row is new (just as getNewRow) and in addition checks to see if the RowObjUpd is not available, which indicates that this has not been committed. Do some double checking if a rowObjUpd is available to ensure that this is the right one.
- This routine defines a temporary, scoped xp macro in order to allow the {get} to be used.

Returns the value of DataModified, indicating whether the current SCREEN-VALUES have been modified but not saved.

Returns DataModified, which will be TRUE if the SmartDataField value has been changed but not saved, otherwise FALSE.

DataModified

Disables searching while there is an update in progress in the SmartDataBrowser. SmartDataBrowser version of setDataModified.

DataObject

The {get} pseudo-function returns the value of DataObject. This value represents the filter used at design time.

DataObjectHandle

Returns the handle of the design-time DataObject.

DataObjectNames

Returns the ordered list of ObjectNames of contained SDOs. This property is normally changed through the SBO Instance Property dialog. It should not be changed until after the Objectnames for the SDOs within the SBO have been set. This function must be run to retrieve the value so that it can check whether the value is still valid, which might not be the case if objects have been removed, added, or replaced since the SBO was last saved. If it no longer matches the list of contained SDOs, then it is blanked out so the default value will be recalculated.

DataObjectOrdering

The {get} pseudo-function returns the value of DataObjectOrdering, the mapping of the order of Update Tables as generated by AppBuilder to the developer-defined update order.

DataQueryBrowsed

Returns the DataQueryBrowsed value after mapping the requesting Browser (or other such client object) to the SDO whose query it is browsing.

Returns TRUE if this SmartDataObject's Query is being browsed by a SmartDataBrowser.

DataQueryString

The {get} pseudo-function returns the value of DataQueryString.

DataSignature

Returns a character string value, which is a list of integers corresponding to the datatypes of the fields in the object temp-table, for use in comparing objects for equivalence.

DataSignature

A character string value that is a list of integers corresponding to the datatypes of the fields in the RowObject temp-table, for use in comparing objects for equivalence.

```
1 = CHARACTER
2 = DATE
3 = LOGICAL
4 = INTEGER
5 = DECIMAL
6 = Reserved for FLOAT OR DOUBLE in the future
7 = RECID
8 = RAW
9 = Reserved for IMAGE in the future
10 = HANDLE
13 = ROWID
```

DataSource

The {get} pseudo-function returns the value of DataObjectHandle.

DataSource

Returns the value of DataSource. To allow design-time override, this does not use the {get} syntax.

DataSourceEvents

The {get} pseudo-function returns the value of DataSourceEvents, a comma-separated list of the events this object wants to subscribe to in its data source.

DataSourceFilter

The {get} pseudo-function returns the value of DataSourceFilter, an optional filter expression for the data source.

DataSourceNames

The {get} pseudo-function returns the value of DataSourceNames, the ObjectName of the Data Object that sends data to this visual object. This would be set if the data source is an SBO or other Container with DataObjects.

DataTarget

The {get} pseudo-function returns the value of DataTarget, a handle in character format, or a comma-separated list of handles for the case of multiple data targets.

DataTargetEvents

The {get} pseudo-function returns the value of DataTargetEvents, the list of events this object class should be subscribed to in its data targets.

DataValue

Returns the data value.

The {get} pseudo-function returns the value of a SmartDataField's DataValue.

Returns the data value that is read by the SmartDataViewer when it calls collectChanges, if it encounters this PROCEDURE in the list of EnabledHandles and the DataModified property of this object equals TRUE.

DBAware

The {get} pseudo-function returns the value of DBAware, TRUE if this object is dependent on being connected to a database. This allows some code, for example in data objects, to execute two different ways.

DBNames

Returns a comma-delimited list of DBNames that corresponds to the Tables in the Query Objects.

DBNames

Returns the value of DBNames, a comma-delimited list of DBNames that corresponds to the tables in the Query Objects

DC

Defined as EXTERNAL "user32"

Allocates a device context for the Treeview

DefaultCharWidth

The {get} pseudo-function returns the value of DefaultCharWidth, the default width of character fields.

DefaultEditorLines

The {get} pseudo-function returns the value of DefaultEditorLines, the default number of inner lines for editors.

DefaultLayout

The {get} pseudo-function returns the value of DefaultLayout.

DefaultLogical

Not currently used. The {get} pseudo-function returns the value of DefaultLogical.

DefaultWidth

The {get} pseudo-function returns the value of DefaultWidth, the default width for noncharacter fields.

DefineAnyKeyTrigger

Returns the DefineAnyKeyTrigger property, TRUE if a persistent trigger is to be defined on ANY-KEY. Does not use the {get} syntax.

DeleteTables

1.) Retrieves a comma-separated list of tables to be deleted.

2.) Stores a comma-separated list of tables to be deleted. Note:

- This can be used to delete only one of the joined tables, it can also be useful to define the sequence of deletion.
- The empty string is treated as a "*": if the argument is a blank, then the deleteRow() will delete all the tables in the query.

DescSubstitute

The {get} pseudo-function returns the value of DescSubstitute.

DesignDataObject

The {get} pseudo-function returns the value of DesignDataObject, the design time SDO for objects that need SDO data, but cannot be linked.

DesignTimeHideMenu

Returns glDesignTimeHideMenu. This flag is set right before initializeobject for a toolbar if this menu is on the startpage.

Destination

The {get} pseudo-function returns the value of Destination.

DestinationList

The {get} pseudo-function returns the value of DestinationList.

Destinations

The {get} pseudo-function returns the value of Destinations.

DestroyStateless

The {get} pseudo-function returns the value of DestroyStateless, which determines whether the persistent SDO should be destroyed on stateless requests.

DeviceCaps

Defined as EXTERNAL "gdi32"

Returns the device caps for the Treeview

DirectionList

The {get} pseudo-function returns the value of DirectionList.

DisabledActions

Returns the DisabledActions property value, a comma-separated list of disabled actions. Placing actions in this list immediately disables them and calls to enableActions have no effect on them while they are listed. That makes it possible to permanently disable actions regardless of state changes. Removing actions from this list with modifyDisabledActions allows enableActions to enable them again.

DisabledAddModeTabs

The {get} pseudo-function returns the value of DisabledAddModeTabs.

DisableOnInit

The {get} pseudo-function returns the value of DisableOnInit, indicating whether the current object should be left disabled when it is first initialized.

DisconnectAppServer

The {get} pseudo-function returns the value of DisconnectAppServer. This value determines whether the persistent SDO disconnects the AppServer.

DisplayDataType

The {get} pseudo-function returns the value of DisplayDataType.

DisplayedField

The {get} pseudo-function returns the value of DisplayedField, the name of the field to display in the selection.

DisplayedFields

The {get} pseudo-function returns the value of DisplayedFields. Used in such activities as signature matching.

DisplayFieldsSecurity

The {get} pseudo-function returns the value of DisplayFieldsSecurity, a comma-separated list of security types corresponding to AllFieldHandles.

DisplayedTables

The {get} pseudo-function returns the value of DisplayedTables. This is the list of SDO table names used by the visualization. Can be only "RowObject". If the object was built against an SBO, it will be the list of SDO ObjectNames whose fields are used.

DisplayField

The {get} pseudo-function returns the value of DisplayField, which will be TRUE if the SmartDataField is to be displayed along with other fields in its Container, otherwise FALSE.

DisplayFormat

The {get} pseudo-function returns the value of DisplayFormat.

DisplayValue

The {get} pseudo-function returns the saved screen/display value of a SmartDataField.

Returns the DisplayValue based on the DataValue. If the keyfield and displayfield are the same, the keyfield value (datavalue) is returned. If they are not the same, then if a reposition source is set, it returns the displayedfieldvalue that is obtained from the SDO. In all other cases it will search the list-item-pairs (or radio-buttons for radio-sets) given the datavalue and will return the display value.

DocumentElement

Returns the ID of the root element.

DocumentHandle

Returns the handle of the current document.

DocumentInitialized

Determines if the document is initialized. For example, it has a root node.

Domain

The {get} pseudo-function returns the value of Domain.

Down

Sets the down attribute for the browse and resizes the browse appropriately. Called from initializeObject for dynamic SmartDataBrowsers.

DTDPublicId

Returns the DTDPublicId property of the Document.

DTDPublicIdList

The {get} pseudo-function returns the value of DTDPublicIdList. That value is a CHR(1)-separated list of DTD Public Ids for producer.

DTDSystemID

Returns the DTDSystemId property of the Document.

DTDSystemIdList

The {get} pseudo-function returns the value of DTDSystemIdList. That value is a CHR(1)-separated list of DTD System Ids for producer.

DynamicObject

The {get} pseudo-function returns the value of DynamicObject.

DynamicSDOProcedure

The {get} pseudo-function returns the value of DynamicSDOProcedure. This value is the name of the dynamic SDO procedure. That is `adm/dyndata.w` by default, but it can be modified if the dynamic SDO is customized.

E

EdgePixels

Returns the number of pixels that should be used to draw the rectangle around the buttons on a SmartPanel.

EdgePixels

The {get} pseudo-function returns the value of EdgePixels.

Editable

The {get} pseudo-function returns the value of Editable. This value indicates whether this object can be edited (add/copy/save/update). If the {get} returns undefined, the routine {get}s EnabledFields. If there are some, the function returns TRUE; if none, the routine calls the same function in all the routines in the GroupAssignTarget list.

EnabledFields

The {get} pseudo-function returns the value of EnabledFields. Can be a comma-separated list of the names of enabled fields in this object that map to fields in the SmartDataObject (&ENABLED-FIELDS).

EnabledHandles

The {get} pseudo-function returns the value of EnabledHandles. This is a comma-separated list of the handles, in character format, of the enabled fields in the visualization object.

EnabledObjFlds

The {get} pseudo-function returns the value of EnabledObjFields, a list of the field names of widgets enabled in this object not associated with data fields.

EnabledObjHdls

The {get} pseudo-function returns the value of EnabledObjHandles, the list of the handles of widgets enabled in this object and not associated with data fields.

EnabledTables

Returns a list of the database tables that have enabled fields.

EnableField

The {get} pseudo-function returns the value of EnableField. This value is TRUE if the SmartDataField is to be enabled for user input along with other fields in its Container, otherwise FALSE.

EnableOnAdd

Not in use at present. The {get} pseudo-function returns the value of EnableOnAdd.

ExitBrowseOnAction

The {get} pseudo-function returns the value of ExitBrowseOnAction, TRUE if the selection of a value in the browse also should Exit the browse.

ExpandOnAdd

The {get} pseudo-function returns the value of ExpandOnAdd.

ExternalJoinList

Sets the ExternalJoinList property (see `webrprop.i`), a comma-separated list of URL parameters that are needed to join an external table (supplied by the ExternalTables URL parameter) to retrieve data for this request. In report and detail Web objects, this is specified on the External Tables and Objects Page in the QueryBuilder. A pipe (|)-separated list of OF phrases. For example:

```
/*This example is from a report web object that is looking for an
  external table of customer, and will present orders, orderlines and
  items based on the ExternalJoinlist as shown.*/

setExternalJoinList('Order.CustNum = Customer.CustNum,OF Order,OF
OrderLine').
```

ExternalRefList

Returns External Ref List property value.

ExternalTableList

Sets the ExternalTableList property (see webrprop.i), a pipe-separated list of comma-separated lists of External Tables that might be used to retrieve data for this request. In report and detail Web objects, this is specified on the External Tables and Objects Page in the QueryBuilder. A pipe (|)-separated list of comma-separated tables. Note: The ExternalJoinList and ExternalWhereList and/or ForeignFieldList have corresponding entries. For example:

```
/* If you have a report of orders called reord.html you might call that
   report from any of the following types of objects:

   objects that provide ExternalTables=customer in the URL, along with
   information (rowid or foreign fields) that allow this report to
   retrieve only orders for a particular customer.

   objects that provide ExternalTables=salesrep,customer in the URL,
   along with information (rowids or foreign fields) that allow this
   report to retrieve only orders for a particular customer sold by a
   particular salesrep.

   objects that provide ExternalTables=salesrep in the URL, along with
   information (rowid or foreign fields) that allow this report to
   retrieve only orders sold by a particular salesrep.

   To allow external joins from any of these types of web objects, you
   would use the following call to setExternalTables: */

   setExternalTableList('customer|salesrep,customer|salesrep').

/* setForeignFieldList, setExternalJoinList and setExternalWhereList
have
   corresponding pipe-delimited entries such as the following: */

setExternalJoinList
   ('OF customer|Order.CustNum = Customer.CustNum AND
   Order.salesrep = Salesrep|Order.salesrep = Salesrep').
   setForeignFieldList ('custnum|salesrep,custnum|salesrep').
   setExternalWhereList('|Order.PromiseDate < 12/12/99|')
```

ExternalTables

Sets the ExternalTables property (see webrprop.i), a comma-separated list of the current List of ExternalTables to use to retrieve data for this request. A comma-separated list of table names.

ExternalWhereList

Sets the ExternalWhereList property (see `webrprop.i`), which contains an optional pipe-separated list of WHERE clause field expressions that correspond to the ExternalTableList Property. A pipe (|)-separated list of field expressions. Note:

- The expressions are defined when the QueryBuilder is called from the External Tables and Object page of Wizards for Embedded SpeedScript objects (report and detail objects).
- WHERE clauses that are set with the "Where" radio-set in the "Edit Join" query builder page on the External Tables and Objects Page of the Report and Detail Wizard are stored in this property.
- WHERE clauses that are in the ExternalJoinList are merged with the clauses in the ExternalWhereList Property. This is useful in cases in which WHERE clauses that reference more than one table need to be built, such as the following:

```
EACH Order WHERE Order.CustNum = Customer.CustNum
      AND Order.SalesRep = Salesrep.salesrep NO-LOCK
      INDEXED-REPOSITION
```

F

FieldColumn

The {get} pseudo-function returns the value of FieldColumn, the column number of the leftmost field.

FieldEnabled

The {get} pseudo-function returns the value of FieldEnabled, TRUE if the SmartDataField is enabled for user input, otherwise FALSE.

FieldFormats

The {get} pseudo-function returns the value of FieldFormats. Internal override of formats for fields.

FieldHandles

The {get} pseudo-function returns the value of FieldHandles. This value is a comma-separated list of the handles, in character format, of the Data Fields in the visualization object.

FieldHelpIds

The {get} pseudo-function returns the value of FieldHelpIds, the internal list of properties for the fields. Each element consists of a name and a value, separated by CHR(1).

FieldHidden

Hides or unhides a lookup field.

FieldLabel

The {get} pseudo-function returns the value of FieldLabel.

FieldLabel

The {get} pseudo-function returns the value of FieldLabel.

FieldLabels

The {get} pseudo-function returns the value of FieldLabels, the internal list of properties for the fields. Each element consists of a name and a value, separated by CHR(1).

FieldName

The {get} pseudo-function returns the value of FieldName, the name of the SDO field to which this object maps.

FieldOperatorStyles

The {get} pseudo-function returns the value of FieldOperatorStyles, an internal list of properties for the fields. Each element consists of a name and a value, separated by a chr(1).

FieldsEnabled

The {get} pseudo-function returns the value of FieldsEnabled. This logical value indicates whether the fields in this visualization object are enabled.

FieldToolTip

The {get} pseudo-function returns the value of FieldToolTip.

FieldToolTips

The {get} pseudo-function returns the value of FieldToolTips, an internal list of properties for the fields. Each element consists of a name and a value, separated by a CHR(1).

FieldWidths

The {get} pseudo-function returns the value of FieldWidths, an internal list of properties for the fields. Each element consists of a name and a value, separated by a CHR(1).

FillBatchOnRepos

The {get} pseudo-function returns the value of FillBatchOnRepos. This value determines whether fetch RowIdent should retrieve enough rows to fill a batch of records when repositioning to the end or near the end of the dataset where an entire batch would not be retrieved.

FilterActive

Returns TRUE if the DataSource has a logical filter

FilterActive

The {get} pseudo-function to extract the values of FilterActive and QueryColumns. Returns TRUE if a filter is active and QueryColumns is not the empty string.

- Enables the {get} on FilterActive using a scoped xp definition.
- It might be set to true explicitly or use the Querycolumns

FilterAvailable

The {get} pseudo-function returns the value of FilterAvailable.

FilterSource

The {get} pseudo-function returns the value of FilterSource. This value represents the Filter Source for pass-through support.

FilterTarget

1.) The {get} pseudo-function returns DataObjectHandle. Returns it in character format.

2.) Returns the value of FilterTarget, the name of the linked filter object. Currently supports only one.

FilterTargetEvents

The {get} pseudo-function returns the value of FilterTargetEvents, the comma-separated list of the events to which this object wants to subscribe to in its FilterTarget.

FilterWindow

The {get} pseudo-function returns the value of FilterWindow, the name of the partition, if any, on which this object will run.

FirstResultRow

The {get} pseudo-function returns the value of FirstResultRow. This will be unknown if the first row has not yet been fetched, otherwise 1 concatenated with the ROWID.

FirstRowNum

The {get} pseudo-function returns the value of FirstRowNum, the temp-table row number of the first row.

FlagValue

The {get} pseudo-function returns the value of FlagValue, the option flag key values for "All" and "None".

FlatButtons

Not in use at present. The {get} pseudo-function returns the value of FlatButtons, TRUE if buttons are shown without obvious borders unless they have the focus.

FolderWindowToLaunch

The {get} pseudo-function returns the value of FolderWindowToLaunch.

ForeignFieldList

1.) Retrieves the alternative list of foreign fields corresponding to the ExternalTableList.

2.) Sets the ForeignFieldList property (see `webrprop.i`), a pipe (|)-separated list of comma-separated lists of the use to format URL parameters to pass to the next Web object.

ForeignFields

1.) Returns the ForeignFields property. That holds the mapping of fields in another DataSource to fields in this SmartDataObject's RowObject temp-table, to open a dependent query.

2.) The {get} pseudo-function returns the value of ForeignFields for the SBO.

ForeignValues

1.) The {get} pseudo-function returns ForeignValues.

2.) The values of the most recently received ForeignField values received by dataAvailable. The values are character strings formatted according to the field format specification and they are separated by the CHR(1) character.

Format

The {get} pseudo-function returns the value of Format, an overridden format string.

FrameHandle

1.) Returns the frame handle of the object.

2.) Store the handle of the frame.

FrameMinHeightChars

The {get} pseudo-function returns the value of FrameMinHeightChars, the predetermined character height of a frame.

FrameMinWidthChars

The {get} pseudo-function returns the value of FrameMinWidthChars, the predetermined character width of a frame.

FullRowSelect

Returns the FullRowSelect property of the tree. If TRUE, the entire node (text and icon) is highlighted.

FuncLibHandle

PRIVATE function. Returns the handle of the AppBuilder function library.

G

GroupAssignSource

The {get} pseudo-function returns the value of GroupAssignSource. This value is the handle of the object's GroupAssign source.

GroupAssignSourceEvents

The {get} pseudo-function returns the value of GroupAssignSourceEvents. This value is a comma-separated list of the events this object wants to subscribe to in its GroupAssign source.

GroupAssignTarget

The {get} pseudo-function returns the value of GroupAssignTarget. This is the handle, in character format, of the object's GroupAssign target.

GroupAssignTargetEvents

The {get} pseudo-function returns the value of GroupAssignTargetEvents. This is a comma-separated list of the events this object wants to subscribe to in its GroupAssign target.

H

Height)

Returns the height of the object.

HelpId

The {get} pseudo-function returns the value of the optional HelpId of the selection.

HiddenActions

The {get} pseudo-function returns the value of HiddenActions, a comma separated list.

HiddenMenuBands

The {get} pseudo-function returns the value of HiddenMenuBands, a comma-separated list.

HiddenToolbarBands

The {get} pseudo-function returns the value of HiddenToolbarBands, a comma-separated list.

HideOnInit

The {get} pseudo-function returns the value of HideOnInit. This value indicates whether the current object should be left hidden when it is first initialized.

HideSelection

Returns the value of HideSelection, TRUE if the current node in the TreeView will not remain highlighted when focus leaves the TreeView.

I

ILComHandle

Returns the com-handle of the ImageList ActiveX object

Image

Loops through the buttons loading their images, if any. The name of the button of interest. The button-image number.

ImageHeight

Returns the ImageHeight property, the height of images in the image list associated with the TreeView.

ImagePath

The {get} pseudo-function returns the value of ImagePath, the path to the images in the filesystem.

ImageWidth

Returns the ImageWidth property, the width of images in the image list associated with the TreeView.

Indentation

Returns the Indentation property of the tree, the number of pixels of indentation between two generations of nodes.

IndexInformation

Returns IndexInormation formatted as the 4 GL index-information attribute, but with RowObject column names, CHR(1) as index separator, and CHR(2) as table separator.

- Intended for internal use by other index info functions, which use this as input to indexInformation(). (Unmapped columns are returned fully qualifed!) This property can be used as input parameter to indexInformation() for further refinement. - If the property is "?", it calls the indexInformation() in query.p and stores the returned value for future calls.
- If a similar list with database fields are needed, use indexInformation() directly (if connected).

InitialPageList

The {get} pseudo-function returns the value of InitialPageList. This value is the comma-delimited list of pages to construct at startup, or "*" to indicate all pages must be initialized at startup.

InMessageSource

The {get} pseudo-function returns the value of InMessageSource, a handle.

InMessageTarget

The {get} pseudo-function returns the value of InMessageTarget.

InnerLines

The {get} pseudo-function returns the value of Inner-Lines, the option flag key values for ‘all’ and ‘none’.

InstanceProperties

The {get} pseudo-function returns the value of InstanceProperties, a list of the ADM instance properties of the SmartObject. Instance properties are those properties that can be set at design time to be initialized as part of startup.

InternalEntries

Returns internal entries of SDO as internal entries cannot be accessed for remote proxy procedures.

InternalRefList

The {get} pseudo-function returns the value of InternalRefList.

J

JMSpartition

The {get} pseudo-function returns the value of JMSpartition.

JMSpassword

The {get} pseudo-function returns the value of JMSpassword.

JMSsession

Returns the handle of the JMS session this instance is using.

JMSuser

The {get} pseudo-function returns the value of JMSuser.

K

KeyDataType

The {get} pseudo-function returns the value of KeyDataType.

KeyField

1. The {get} pseudo-function returns the value of KeyField, which is the name of the key field.

2.) That is the name of the field whose value is:

- Received from the SmartDataViewer in setDataValue
- Retrieved by the SmartDataViewer in getDataValue

KeyFields

The {get} pseudo-function returns the value of KeyFields. This value is a comma-separated list.

KeyFieldValue

The {set} pseudo-function assigns a new value to KeyFieldValue. Enables the {set} using a temporary scoped macro.

KeyFormat

The {set} pseudo-function assigns a new value to KeyFormat.

L

Label

The {get} pseudo-function returns the value of Label, the label defined for the selection. If the value is undefined, it is converted to a string. Because no global xp macro is defined, this function defines a temporary, scoped xp macro.

LabelEdit

Returns the LabelEdit property of the tree. If TRUE, the user can click on the node to edit it.

LabelHandle

The {get} pseudo-function returns the value of LabelHandle.

LastDbRowIdent

The {set} pseudo-function sets a new value for LastDbRowIdent. This value will be unknown if the last row has not been fetched, otherwise it will be the database rowid(s) for the last row.

LastResultRow

The {get} pseudo-function returns the value of LastResultRow. This value will be unknown if the last row has not yet been fetched, otherwise RowNum is concatenated with ROWID.

LastRowNum

The {get} pseudo-function returns the value of LastRowNum, the temp-table row number of the last row.

LayoutOptions

The {get} pseudo-function returns the value of LayoutOptions, the list of multi-layout options for the object.

LayoutVariable

The {get} pseudo-function returns the value of LayoutVariable, the name of the &LAYOUT-VARIABLE preprocessor for the object, which is used as a prefix to the name of the procedure that resets it.

LineStyle

Gets the LineStyle property of the tree:

- 0 (Default) — Displays lines between child nodes and their parent.
- 1 — Displays lines between root nodes as well as lines between child and parent nodes.

LinkColumns

Stores the columns that have hyperlinks as a comma-separated list in the LinkURLs property (see `wbtaprop.i`). The comma-separated list of columns. For example:

```
setLinkColumns ("custnum,salesrep")
```

LinkedFieldDataTypes

The {get} pseudo-function returns the value of LinkedFieldDataTypes.

LinkedFieldFormats

The {get} pseudo-function returns the value of LinkedFieldFormats.

ListItemPairs

The {get} pseudo-function returns the value of ListItemPairs.

LoadedByRouter

The {get} pseudo-function returns the value of LoadedByRouter. That value is true if the XML and Schema have already been loaded by the router.

LogFile

The {get} pseudo-function returns the value of LogFile.

LogicalObjectName

The {get} pseudo-function returns the value of LogicalObjectName. Defines a temporary, scoped xp macro for the purpose.

LogicalVersion

The {get} pseudo-function returns the value of LogicalVersion. Defines a temporary, scoped xp macro for the purpose.

LogicBuffer

Returns the handle of the data-logic table, if possible.

LogicalObjectName

Calls SUPER with the argument, and then uses {get} to check ObjectInitialized. If that value is true, this routine clears it and runs initializeObject. This routine always returns TRUE.

LookupFilterValue

Returns gcLookupFilterValue, the filter value set.

LookupHandle

The {get} pseudo-function returns the value of LookupHandle.

LookupImage

The {get} pseudo-function returns the value of LookupImage.

LookupQuery

This routine is published from the viewer and is used to pass the query required by this lookup back to the viewer for building. Once built, the query will be returned into the procedure displayLookup.

- This routine is designed to facilitate all lookup queries being built with a single AppServer hit. It is published from displayfields in the viewer.
- This routine is not run at all in add mode.

M

MaintenanceObject

The {get} pseudo-function returns the value of MaintenanceObject.

MaintenanceSDO

The {get} pseudo-function returns the value of MaintenanceSDO.

ManualAddQueryWhere

The {get} pseudo-function returns the value of ManualAddQueryWhere. This value is reapplied by the filter when needed to ensure the original query remains intact.

- Value is pcwhere + CHR(3) + pcbuffer or empty or " ? " + CHR(3) + pcandor
- Multiple entries are supported, separated by CHR(4).

ManualAssignQuerySelection

The {get} pseudo-function returns the value of ManualAssignQuerySelection. This value is reapplied by the filter when needed to preserve the original query.

- Value is pccolumns + CHR(3) + pcvalues + CHR(3) + pcoperators.
- Multiple entries are supported, separated by chr(4).

ManualSetQuerySort

The {get} pseudo-function returns the value of ManualSetQuerySort. This value is reapplied by the filter when needed to preserve the original query.

MapNameProducer

The {get} pseudo-function returns the value of MapNameProducer.

MapObjectProducer

The {get} pseudo-function returns the value of MapObjectProducer.

MapTypeProducer

The {get} pseudo-function returns the value of MapTypeProducer.

MarginPixels

The {get} pseudo-function returns the value of MarginPixels, the number of pixels to reserve for the Panel margin used by resizeObject.

MasterDataObject

The {get} pseudo-function returns the value of MasterDataObject, the handle of the SDO that has no data source of its own and is the parent to other SDOs.

MaxWidth

The {get} pseudo-function returns the value of MaxWidth. That value is the maximum width of the browse when CalcWidth is TRUE.

Menu

The {get} pseudo-function returns the value of Menu, TRUE if a menu is to be generated.

MenuMergeOrder

The {get} pseudo-function returns the value of MenuMergeOrder. This value decides the order in which menus will be merged with other toolbar instances.

MessageType

The {get} pseudo-function returns the value of MessageType.

MinHeight

The {get} pseudo-function returns the value of MinHeight.

MinWidth

The {get} pseudo-function returns the value of MinWidth.

MultiInstanceActivated

The {get} pseudo-function returns the value of MultiInstanceActivated.

MultiInstanceSupported

The {get} pseudo-function returns the value of MultiInstanceSupported.

N

NameList

The {get} pseudo-function returns the value of NameList.

NameSpaceHandle

The {get} pseudo-function returns the value of NameSpaceHandle. That value is the handle of the loaded XML mapping schema namespaces.

NavigationSource

1.) The {get} pseudo-function returns the value of NavigationSOurce, the handle of the query object's Navigation source.

2.) This value is used for pass-through for regular containers, and by the SBO generally.

NavigationSourceEvents

The {get} pseudo-function returns the value of NavigationSourceEvents. This is the list of events to be subscribed to in the Navigation Panel or other Navigation-Source.

NavigationTarget

The {get} pseudo-function returns the value of NavigationTarget.

NavigationTargetEvents

The {get} pseudo-function returns the value of NavigationTargetEvents, a comma-separated list of the events this object wants to subscribe to in its NavigationTarget.

NavigationTargetName

The {get} pseudo-function returns the value of NavigationTargetName. This is the ObjectName of the Data Object to be navigated by this panel. This would be set if the Navigation-Target is an SBO or other Container with DataObjects.

NewNode

1.) Returns TRUE if the current RowObject record is in new mode. Returns "?" if there is no current RowObject.

2.) Returns TRUE if the matching contained SDO is in NewMode. This is the SBO version of getNewMode.

3.) Returns the next unused node number in sequence.

NewRecord

The {get} pseudo-function returns the value of NewRecord. This is a character value that indicates whether the current record in the visualization object is newly created. Valid values are: "Add", "Copy", and "No".

NewRow

1.) Returns TRUE if the current RowObject record is new (an added record or a copy of an existing record has never been written to the database). Returns "?" if there is no current RowObject.

2.) his SBO version of getDataHandle is run from a browser to get the query from the contained Data object.

NextNodeKey

Returns a unique code to be used for the Key attribute of the Treeview.

NodeExpanded

Expands or collapse a node according to the argument passed.

NumDown

The {get} pseudo-function returns the value of NumDown. That value is the number of rows that are displayed DOWN in the browse.

NumRows

The {get} pseudo-function returns the value of NumRows, the number of rows to display in the selection widget.

O

ObjectEnabled

Returns a flag to indicate whether the current object is enabled.

ObjectHidden

The {get} pseudo-function returns the value of ContainerHidden or ObjectHidden, indicating whether the current object is hidden. Before checking ObjectHidden, the routine examines the parent state in ContainerHidden. If ContainerHidden is true, there is no need to examine ObjectHidden, and the function immediately returns TRUE. Note that "hidden" is a logical concept in the ADM. A nonvisual object can be "hidden" to indicate that it is not currently active in some way, because it is a Container-Target of some visual object that is hidden.

ObjectInitialized

The {get} pseudo-function returns the value of ObjectInitialized, TRUE if this object has been initialized.

ObjectLayout

The {get} pseudo-function returns the value of ObjectLayout, the current Layout Name for the object.

ObjectMapping

The {get} pseudo-function returns the value of ObjectMapping. This value is a list of handles of Navigation-Source objects (panels) or other objects that are mapped to contained Data Objects, and the SDOs the SBO has connected them up to, according to their NavigationTargetName property or setCurrentMappedObject request.

ObjectMode

Returns the mode of the object. Valid values are:

- 'Modify' — This is a non-modal mode that can be turned off with view
- 'Update' — This is a modal mode that is turned off with save or cancel
- 'View'— This is not editable

ObjectName

The {get} pseudo-function returns the value of ObjectName. This value is the name of the object, which can be the filename or some other designation meaningful to the repository and other objects.

ObjectPage

The {get} pseudo-function returns the value of ObjectPage, the logical page on which this object has been placed.

ObjectParent

1.) Returns a handle value of the Progress parent container handle for the current visualization object.

2.) Returns the widget handle of the caller's **parent's** parent (that is: the container that is the caller's logical grandparent), if any.

ObjectType

The {get} pseudo-function returns the value of ObjectType, unless {&ADMSuper} is nonempty. This value is the type of the SmartObject, such as "SmartDataObject", etc. "SUPER" is returned in the case of a super procedure.

ObjectVersion

The {get} pseudo-function returns the value of ObjectVersion, the ADM version of the SmartObject.

OLEDrag

Returns the value of OLEDrag, TRUE if drag is supported.

OLEDrop

Returns the value of OLEDrop, TRUE if drop is supported.

OpenOnInit

The {get} pseudo-function returns the value of OpenOnInit, which determines whether queries should be opened at initialization.

OpenQuery

1.) Returns the original design WHERE clause for the database query.

2.) Returns the value of OpenQuery, to allow it to be manipulated by setQueryWhere, for example.

- This is important because no matter how the query is dynamically modified, it can be reset to its original state using this return value in a QUERY-PREPARE method.
- getOpenQuery is called from the client on AppServer, so do not use the xp preprocessor.

Operator

The {get} pseudo-function returns the value of Operator, the default when OperatorStyle is "Implicit".

OperatorLongValues

The {get} pseudo-function returns the value of OperatorLongValues, a list of operators and long text.

OperatorStyle

The {get} pseudo-function returns the value of OperatorStyle. There are four valid values:

- "Explicit" – Specify operator in a separate widget
- "Implicit" – Use the Operator and UseBegins property
- "Range" – Use two fill-ins and specify GE and LE values
- "Inline" – Type the operator into the field (defaults to Equals, or BEGINS if UseBegins is TRUE).

OperatorViewAs

The {get} pseudo-function returns the value of OperatorViewAs. This value is the view-as type used to define the operator when OperatorStyle equals "Explicit".

Optional

The {get} pseudo-function returns the value of Optional. The value will be TRUE if the selection is optional, and in that case the property OptionalString holds the value to display.

OptionalBlank

The {get} pseudo-function returns the value of OptionalBlank, TRUE if the optional value is a blank value. Applies to character fields only.

OptionalString

The {get} pseudo-function returns the value of OptionalString. This is displayed as an optional value when the Optional property is set to TRUE.

OutMessageSource

The {get} pseudo-function returns the value of OutMessageSource.

OutMessageTarget

The {get} pseudo-function returns the value of OutMessageTarget.

P

PageNTarget

The {get} pseudo-function returns the value of PageNTarget. This is the list of objects that are on some page other than 0. The list items are of the form handle|pagenum. Use addLink rather than editing them by hand.

PageSource

The {get} pseudo-function returns the value of PageSource. This is the handle of the object's folder, if any.

PanelFrame

The {get} pseudo-function returns the value of PanelFrame. This is the Frame handle of the SmartPanel, for resizeObject.

PanelLabel

The {get} pseudo-function returns the value of PanelLabel. This is the handle of the Panel's Label, if any.

PanelState

Returns the value of PanelState, the current state of the SmartPanel. The {get} syntax is not used here.

PanelType

The {get} pseudo-function returns the value of PanelType, the type of SmartPanel ('Update', 'Navigation')

Parent

Returns some parent

ParentDataKey

The {get} pseudo-function returns the value of ParentDataKey.

ParentField

The {get} pseudo-function returns the value of ParentField.

ParentFilterQuery

The {get} pseudo-function returns the value of ParentFilterQuery.

PassThroughLinks

Returns the list of link types that can be pass-through links. Used by addLink. This property value is stored in `smart.p` for all SmartObjects.

Persistency

The {get} pseudo-function returns the value of Persistency.

PhysicalObjectName

The {get} pseudo-function returns the value of PhysicalObjectName.

PhysicalVersion

The {get} pseudo-function returns the value of PhysicalVersion.

PingInterval

The {get} pseudo-function returns the value of PingInterval.

PrimarySdoTarget

The {get} pseudo-function returns the value of PrimarySdoTarget.

Priority

The {get} pseudo-function returns the value of Priority.

PromptLogin

The {get} pseudo-function returns the value of PromptLogin.

Property

Returns the specified property for some node:

- BACKCOLOR — The background color of the node
- BOLD — Whether the node is bold
- CHECKED — Whether the node is checked
- CHILD — The first child node
- CHILDREN — The number of children of the node
- COUNT — The number of nodes in the tree
- EXPANDED — Whether the node is expanded or collapsed
- FIRSTSIBLING — The first sibling of the node
- FORECOLOR — The foreground color of the note
- FULLPATH — The full path of the node
- GETVISIBLECOUNT — The number of visible nodes
- IMAGE — The image (contained in imagelist) for the node
- INDEX — The index of the node
- KEY — The unique key of the node
- LASTSIBLING — The last sibling of the node
- PARENT — The parent of the node
- NEXT — The next node key
- PATHSEPARATOR — The delimiter returned by the FullPath property
- PREVIOUS — The previous node key
- ROOT — The root of the node
- SELECTEDITEM — The selected node
- SORTED — Whether the children of the node are sorted

- TAG — The tag of the specified node.
- TEXT — The label text of the specified node
- VISIBLE — Whether the entire tree is visible/invisible

PropertyDialog

The {get} pseudo-function returns the value of PropertyDialog, the name of the dialog procedure that sets InstanceProperties.

PropertyList

Sets a list of properties taken from a CHR(3)-delimited list of “prop CHR(4) value” pairs.

Q

QueryColumns

The {get} pseudo-function returns the value of QueryColumns.

QueryContainer

The {get} pseudo-function returns the value of QueryContainer. This value indicates whether the Container is itself a QueryObject. This routine is used to determine if an SBO handles the transaction for.

QueryContext

Internal function, developed for use in generic calls. The {get} pseudo-function returns the value of QueryContext on the client.

QueryEmpty

Checks if the data source or query is empty.

QueryHandle

The {get} pseudo-function returns the value of QueryHandle.

QueryObject

Returns a flag that indicates whether this object manages its own database query.

QueryOpen

Returns TRUE if the Query Object’s database query is currently open.

QueryPosition

1.) The QueryPosition property, valid values for which are "FirstRecord", "LastRecord", "NotFirstOrLast", and "NoRecordAvailable".

2.0) The SBO version of the {get} pseudo-function returns the value of QueryPosition. It uses {get} to determine whether the requester is the target-procedure or the source-procedure, {get}s ObjectMapping value, checks whether the MasterDataObject is involved, and finally returns the value of QueryPosition from the SDO to which the caller is mapped.

QueryRowIden

The {get} pseudo-function returns the value of QueryRowIdent, a RowIdent to be used to position an SDO query when it is first opened.

QueryRowObject

The {get} pseudo-function returns the value of QueryRowObject. That value is the handle of the RowObject temp-table associated with the Browse's query.

QuerySort

Returns the sort phrase.

QueryString

The {get} pseudo-function returns the value of QueryString, the attribute used as working storage for all query manipulation methods.

- The method always returns a WHERE clause. If the QueryString property has not been set, it uses the current WHERE clause - QueryWhere. If there is no current use the design where clause - OpenQuery.
- The openQuery will call prepareQuery with this property.

QueryTables

The {get} pseudo-function returns the value of QueryTables.

QueryWhere

1.) The current query string, returns "?" if the query handle is undefined.

2.) Gets the current WHERE clause for the query.

3.) Prepares the query with a new OPEN QUERY statement or a new expression. The new WHERE clause or expression.

R

RebuildOnRepos

The {get} pseudo-function returns the value of RebuildOnRepos, the flag indicating whether the RowObject temp-table should be rebuilt if a fetchLast or other reposition is done s outside the bounds of the current dataset.

RecordState

The {get} pseudo-function returns the value of RecordState. This is a string indicating whether a record is available or not. Valid values are "RecordAvailable" and "NoRecordAvailable".

ReEnableDataLinks

The {get} pseudo-function returns the value of GetReEnableDataLinks.

Refresh

Refreshes the treeview.

RefreshList

Returns a comma-separated list of the names of the Dynamic Combos to be refreshed for the parent Combo named in the argument.

ReplyReqList

The {get} pseudo-function returns the value of ReplyReqList.

ReplyRequired

The {get} pseudo-function returns the value of ReplyRequired.

ReplySelector

The {get} pseudo-function returns the value of ReplySelector.

ReplySelectorList

The {get} pseudo-function returns the value of ReplySelectorList.

RepositionDataSource

The {get} pseudo-function returns the value of RepositionDataSource, TRUE if the data-source is to be repositioned on VALUE-CHANGED of the select. This is not needed for the view-as browse option. This must be set to TRUE if, for example, the data-source also is a data-source for other objects, and those objects need to be refreshed when a value is changed in the combo-box.

ResizeHorizontal

The {get} pseudo-function returns the value of ResizeHorizontal, TRUE if the object can be resized in X.

ResizeVertical

The {get} pseudo-function returns the value of ResizeVertical, TRUE of the object can be resized in Y.

RootNodeParentKey

Returns the value (from xcRootNodeParentKey) to use as the ParentKey for root nodes on the tree. By searching for nodes with this value as the parent all the root nodes can be found.

RouterSource

The {get} pseudo-function returns the value of RouterSource.

RouterTarget

The {get} pseudo-function returns the value of RouterTarget. The {set} pseudo-function sets a new value for RouterTarget.

Row

Returns the height of the object. Use repositionObject to set the Row.

RowIdent

1.) The value of RowIdent.

2.) The comma-delimited character string containing the ROWIDs of the database records that are the source of the RowObject record.

3.) The ROWID, in character format, of the current row of the visualization. If the update-target is an SDO, the database ROWIDs can be stored as the second entry of the list. If connected to an SBO, the value is a semicolon-separated list corresponding to the SBO's DataObjectNames list. If the SBO is a valid UpdateTarget, then the ROWIDs are for the UpdateTargetNames; otherwise the DataSourceNames. Note: Return only the ROWIDs that uniquely identify this object's connection to the dataSource/updateTarget and remove unnecessary ROWIDs so the property can be used directly as input to update methods in the SBO. The assumption is that all tables that are displayed only in the visual object are on the "one" side of a one-to-many or many-to-one relation of the table that is updated, so that they are uniquely identified through the updatable table and their ROWIDs are not part of this object 'Ident'. However, return more than one ROWID is returned for the case where more than one SDO is updated as one-to-one in the SBO.

Rowids

Gets the ROWID(s) of the current row or rows.

RowObject

1.) The value of RowObject, the handle of the RowObject Temp-Table buffer.

2.) The handle to the Browser's temp-table definition.

RowObjectState

1.) Signals whether there are uncommitted updates in the object.

2.) The value of RowObjectState, which signals whether there are uncommitted updates in the object. Valid return values are 'NoUpdates' and 'RowUpdated'. Does not use the {get} syntax.

RowObjectTable

The temp-table handle of the RowObject table. Note:

- This is not the same as the RowObject property, which is the handle of the RowObject Buffer.
- Does not use the {get} syntax because the setRowObjectTable function must also do other work.

RowObjUpd

The {get} pseudo-function returns the value of the RowObjUpd property. This value is the handle of the temp-table buffer where updates are stored.

RowObjUpdTable

The {get} pseudo-function returns the value of RowObjUpdTable property. This value is the handle of the RowObjUpd temp-table.

RowsToBatch

The value of RowsToBatch, the number of rows to be transferred from the database query into the RowObject temp-table in a single operation.

RunAttribute

The {get} pseudo-function returns the value of RunAttribute.

RunDOOptions

The {get} pseudo-function returns the value of RunDOOptions. This is a comma-separated list with options that determine how Data Objects are run from constructObject The options available are:

- dynamicOnly – Runs dynamic data objects only and supersedes all other options.
- sourceSearch – Searches for source code if r-code is not found.
- clientOnly – Runs proxy (_cl) code only (for both r-code and source code).

If dynamicOnly and/or clientOnly options are used, the container assumes that the Data Object has an AppServer partition defined. If an AppServer partition is not defined and dynamicOnly and/or clientOnly options are used, errors will occur as if the Data Object was being run on an AppServer that had not been started.

RunMultiple

The {get} pseudo-function returns the value of RunMultiple.

S

SavedContainerMode

The {get} pseudo-function returns the value of SavedContainerNode.

SavedScreenValue

Obsolete function replaced by getDisplayValue (which it calls). Otherwise, kept for backwards compatibility.

SaveSource

The {get} pseudo-function returns the value of SaveSource. This value is TRUE if the TableIO-Source is ‘Save’ but FALSE if (modal) ‘Update’.

SchemaHandle

The {get} pseudo-function returns the value of SchemaHandle, the handle of the loaded XML mapping schema

SchemaList

The {get} pseudo-function returns the value of SchemaList.

SchemaManager

Starts the schema manager, if necessary, and returns its procedure handle.

Scroll

Gets the Scroll property of the tree. If TRUE, scrollbars appear.

ScrollRemote

The {get} pseudo-function returns the value of ScrollRemote.

SDFFileName

The {get} pseudo-function returns the value of SDFFileName.

SDFTemplate

The {get} pseudo-function returns the value of SDFTemplate.

SdoForeignFields

The {get} pseudo-function returns the value of SdoForeignFields.

SearchColumns

1.) Contains a comma-delimited list of columns that are used to search in the current data-source. Currently supports only one column.

2.) Stores the SearchColumns value. (Currently one.) A column name in the data-source.

SearchField

The {get} pseudo-function returns the value of SearchField. That value is the name of a field where searching is enabled. If set, space is allocated for a fill-in to accept a value to be repositioned to.

Secured

The {get} pseudo-function returns the value of Secured, TRUE if the combo's security is set to HIDDEN.

SecuredTokens

The {get} pseudo-function returns the value of SecuredTokens, a comma-separated list.

SelectedNode

Returns the key of the selected node.

Selectors

The {get} pseudo-function returns the value of Selectors.

ServerConnection

1.) Returns SESSION:SERVER-CONNECTION-ID through ServerConnection property.

2.) Sets SERVER_CONNECTION_ID property from SESSION:SERVER-CONNECTION-ID.

ServerFileName

Returns the property value representing the actual server-side SDO filename to run on the AppServer. That value might not be the ObjectName if that has been modified.

- No xp macro defined for this property.
- Defaults to the target-procedure filename without the _cl.

ServerOperatingMode

The {get} pseudo-function returns the value of ServerOperatingMode. This value is a string representing the connection state. Valid values are "stateless", "state-reset", "state-aware", or "none" (no AppServer connection).

ServerSubmitValidation

Returns the value of ServerSubmitValidation, which signals whether the column and RowObject Validation procedures done as part of client validation are to be executed on the server side. If * yes * , normally when the SDO is being run through the open client interface, then serverCommit will execute SubmitValidation itself.

ShowBorder

The {get} pseudo-function returns the value of ShowBorder, TRUE if a three-dimensional border is to be used around the buttons and as a delimiter when "RULE" is specified in createToolbar.

ShowCheckBoxes

Returns the value of ShowCheckBoxes, TRUE if there will be check boxes beside each node on the TreeView.

ShowRootLines

Returns the value of ShowRootLines, TRUE if there should be lines leading to the roots of the tree or not.

ShutDownDest

The {get} pseudo-function returns the value of ShutDownDest. This value is the queue to which a message can be sent to shut down a SmartConsumer running unattended.

SingleSel

Returns the value of SingleSel, TRUE if the node is to be expanded when selected or clicked.

Sort

The {get} pseudo-function returns the value of Sort, TRUE if the selection is to be sorted.

StartBrowseKeys

The {get} pseudo-function returns the value of StartBrowseKeys, the list of Keylabels or KeyFunctions that starts the browse.

StaticPrefix

Returns the prefix used before the action name in static definitions

SubModules

The {get} pseudo-function returns the value of SubModules.

Subscriptions

The {get} pseudo-function returns the value of Subscriptions.

SupportedLinks

The {get} pseudo-function returns the value of SupportedLinks, a comma-separated list of the SmartObject links supported by this object.

SupportedMessageTypes

The {get} pseudo-function returns the value of SupportedMessageTypes.

T

TableIOButtons

Convert a tableIO recordState to setButton.

TableIOSource

The {get} pseudo-function returns the value of TableIOSource.

TableIOSourceEvents

The {get} pseudo-function returns the value of TableIOSourceEvents. This is a comma-separated list of the events to which this object wants to subscribe to in its TableIO source.

TableIOTarget

The {get} pseudo-function returns the value of TableIOTarget, a list of the handles, in character format, of the object's TableIO Targets.

TableiOTargetEvents

The {get} pseudo-function returns the value of TableIOTargetEvents, a comma-separated list of the events to which this object wants to subscribe to in its TableIO Target.

TableIOType

1.) The value of TableIOType. A comma-separated list in CHARACTER format of the object's TableIO-Target handles.

2.) tableio-specific values of PanelType, if any, so that resetTabelio can handle panels similar to toolbar.

TableModifier

Stores the Specified HTML attributes for the <table> tag in TableModifier property (see `wbtaprop.i`). One or more HTML attributes that will be used in the TABLE tag. The Embedded SpeedScript (.htm) files created by the Report wizard show examples of this function call in use. For example:

```
setTableModifier('  border="2" cellspacing=10')
```

TableRowids

Retrieves a list of current rowids by using getRowids, and then uses the ExternalTables property (see `webrprop.i`) to remove the External Tables from the list, returning a list of rowids for only the records requested by the current object. Used to pass as external rowids to a called object.

TableRows

1.) Returns the number of rows in the table.

2.) Stores the specified number of rows for the HTML table in TableRows property (see `wbtaprop.i`). Number of rows to display in the HTML page. The Embedded SpeedScript (.htm) files created by the Report wizard show examples of this function call in use. For example:

```
setTableRows(10)
```

Tables

1.) Returns the value of Tables, a comma delimited list of tables in the Query Objects

- Qualified with database name if the query is defined with dbname.
- From Version 9.1B, this property is a design time property while it earlier was resolved from the actual query. This was very expensive as the function then always needed to be resolved on the server and it is rather extensively used.
- There is currently no way to change the order of the tables at run time. But it would be even more important to have this as a property if the order of the tables were changed dynamically, because several other properties have table delimiters and are totally dependent on the design time order of this property.
- This does not have an xpTables preprocessor because some web2 objects override getTables().

2.) Overrides to use the property for HTML mapping with no data-source.

3.) Returns a comma-delimited list of tables from which data will be retrieved, either from a SmartDataObject or database.

TargetNameSpace

Returns the XML Schema TargetNameSpace. This defines the XMLNS attribute of the document instance.

TargetProcedure

Used in many SmartObjects to return the source-procedure’s target-procedure to a function, such as addRow, in an SBO that needs to know who the "real" caller object is., it can

TimeToLive

TimeToLive represents the time during which the message is considered current and not stale.

Toolbar

The value of Toolbar is TRUE if the toolbar is to be created.

ToolbarAutoSize

The value of ToolbarAutoSize is TRUE if the toolbar should be auto-sized to the width of the window at runtime.

ToolbarBands

The {get} pseudo-function returns the value of ToolbarBands, the list of the toolbar bands selected in the Instance Properties.

ToolbarDrawDirection

The {get} pseudo-function returns the value of ToolbarDirection, the draw direction ("horizontal" or "vertical") of the toolbar.

ToolbarHeightPxl

Calculates and returns the toolbar height in pixels from the three properties: ToolHeightPXL, ToolbarMarginPxl, and ShowBorder.

ToolbarInitialState

The {get} pseudo-function returns the value of ToolbarInitialState, the state selected in the Instance Properties. Used to view or hide buttons and menus according to state.

ToolbarParentMenu

The {get} pseudo-function returns the value of ToolbarParentMenu, the menu selected in the Instance Properties. Required only if one or more toolbar menus need to be added under a specific submenu. If the submenu does not exist, it is created.

ToolbarSource

The value of ToolbarSource, a handle or list of handles.

ToolbarSourceEvents

The {get} pseudo-function returns the value of ToolbarSourceEvents. This value is the list of events to be subscribed to in the Toolbar-Source.

ToolbarTarget

The {get} pseudo-function returns the value of ToolbarTarget, a handle.

ToolbarTargetEvents

The {get} pseudo-function returns the value of ToolbarTargetEvents, the list of events to which this object wishes to subscribe in the Toolbar-Target.

ToolbarWidthPxl

The {get} pseudo-function to obtain the values of ToolMarginPxl, ToolWidthPxl, and ToolMaxWidthPxl. Calculates and returns the toolbar width.

ToolMarginPxl

Returns a constant zero at present.

ToolTip

The {get} pseudo-function returns the value of tooltip, the optional tooltip for the selection.

TopOnly

Returns the value of the TOP-ONLY attribute.

TranslatableProperties

The {get} pseudo-function returns the value of TranslatableProperties, a comma-separated list.

TreeDataTable

1.) The handle of a dynamic temp-table populated with property fields for a Treeview.

2.) The value of TreeDataTable, the handle of the temp-table into which tree-population data will be stored.

TreeStyle

Valid values are:

- 0 Text only.
- 1 Image and text.
- 2 Plus/minus and text.
- 3 Plus/minus, image, and text.
- 4 Lines and text.
- 5 Lines, image, and text.
- 6 Lines, plus/minus, and text.
- 7 (Default) Lines, plus/minus, image, and text.

TVControllerSource

The {get} pseudo-function returns the value of TVControllerSource, the handle of the procedure controlling this TreeView.

TVControllerTarget

The {get} pseudo-function returns the value of TVControllerTarget.

TVControllerTargetEvents

The {get} pseudo-function returns the value of TVControllerTargetEvents. This is the list of target events to which this controller will subscribe.

TypeName

The {get} pseudo-function returns the value of TypeName.

U

UIBMode

Returns the UIBMode property to indicate whether this object is in Design mode in AppBuilder (AppBuilder was originally called User-interface Builder). This will return blank if the object is not in design mode, that is, not running in an AppBuilder design window. It will return "Design" if in design mode, or "Design-Child" if it is contained in another SmartObject that is in design mode, such as a SmartFrame. It will return "?" if the object is not a SmartObject and does not have a valid handle in ADM-DATA.

UpdatableColumns

1.) A comma-delimited list of the Updatable Columns for this SmartDataObject.

2.) A list of all Updatable Columns of contained DataObjects, qualified by their ObjectNames.

UpdateActive

The {get} pseudo-function returns the value of UpdateActive, TRUE if ANY of the contained objects have active updates.

UpdateFromSource

The {get} pseudo-function returns the value of UpdateFromSource, TRUE if this object should be updated as one-to-one of the datasource updates.

UpdateMode

1.) Returns the value of the updateMode flag, which governs the behavior of ColumnStringValue. If updateMode is set to “Add”, ColumnStringValue returns initial values of a column. If updateMode is not set to “Add”, ColumnStringValue returns the current value of the column.

2.) Sets the value of the updateMode flag. Either “yes” or “no”. This call is used in processWebRequest in web objects created using the report and detail wizards. In these objects, the hidden field AddMode is set to the value of UpdateMode when a page is sent to the WebBrowser, which is detected when the WebPage is returned.

UpdateSource

1.) The {get} pseudo-function returns the value of UpdateSource. This is used for pass-through links, to connect an object inside the container with an object outside the container. It is CHARACTER because at least one type of container (SBO) supports multiple update sources.

2.) The value of UpdateSource, the handle of the object's update-source.

UpdateTarget

The {get} pseudo-function returns the value of UpdateTarget. This is used for pass-through links, to connect an object inside the container with an object outside the container.

UpdateTargetNames

The {get} pseudo-function returns the value of UpdateTargetNames. This is the ObjectName of the Data Object to be updated by this visual object. This is set if the Update-Target is an SBO or other Container with DataObjects.

UseBegins

The {get} pseudo-function returns the value of UseBegins. This value is true when BEGINS is supposed to be used for character tests.

UseColumnLabels

Stores a logical specifying whether to use column labels for the HTML table in UseColumnLabels property (see `wbtaprop.i`). This is using the LABEL and not the COLUMN–LABEL of the field. (There is no logic to take care of the ! in column–labels). For example:

```
setUseColumnLabels(no)
```

UseContains

The {get} pseudo-function returns the value of UseContains. This value is true when Contains is supposed to be used as operator for character values.

UseDBQualifier

Returns TRUE if table references are qualified with database.

UseDTD

Returns TRUE if DTD is to be used.

UsePairedList

Checks to see if an object is an editable combo-box. If getUsePairedList determines the object is an editable combo-box, it returns FALSE, indicating a paired list cannot be used.

UserProperty

Retrieves the value of a dynamically defined property.

- In addition to properties for which storage in the property temp-table is defined, in addition to specific *get* and *set* functions, it is possible for application code to define a property at run time simply by giving it a name and a value. This function returns the value of such a property.
- LIMITATION: You cannot set or get user-defined properties in an application running in n-tier (client and AppServer) mode.
- Dynamically defined properties are currently stored in a list in the ADM–DATA procedure attribute, delimited by CHR(3), with CHR(4) between property name and value. However, it should not be necessary for developers to be aware of the specific storage mechanism for dynamically defined properties, which could be subject to change in the future.

```
/*
** This example retrieves the current value of a property defined
** dynamically by setUserProperty(). Note that all dynamic
** properties are returned in character format.
*/

iProcessState = INTEGER(getUserProperty('ProcessState':U)).
```

UseRepository

Returns TRUE iff Dynamics is running and the Repository is therefore available.

UserNameA

Returns some parent

V

ValidateOnLoad

The {get} pseudo-function returns the value of ValidateOnLoad: whether the document should be validated on load.

ValidKey

Returns TRUE if some key is valid

ViewAs

The {get} pseudo-function returns the value of ViewAs, the ViewAs definition of the selection: combo-box, radio-set, selection-list, or browse. Uses a colon as a separator to define SUB-TYPE for combo-box or horizontal/vertical radio-set. For example: ’radio-set:vertical’.

ViewAsFields

Not currently used. The {get} pseudo-function returns the value of ViewAsFields.

ViewerLinkedFields

The {get} pseudo-function returns the value of ViewerLinkedFields.

ViewerLinkedWidgets

The {get} pseudo-function returns the value of ViewerLinkedWidgets.

VisibleRowids

The {get} pseudo-function returns the value of VisibleRowids.

VisibleRowReset

The {get} pseudo-function returns the value of VisibleRowReset.

VisualBlank

The {get} pseudo-function returns the value of VisualBlank. This value is used to visualize searching for BLANK values.

W

WaitForObject

The {get} pseudo-function returns the value of WaitForObject. This value is the handle of the object (most likely a SmartConsumer) in the container that contains a wait-for that needs to be started with startWaitFor.

Waiting

The {get} pseudo-function returns the value of Waiting. That value determines when waitForMessages is satisfied.

WebState

1.) Returns a Web object's Web state. Possible values are state–aware, state–less, timed–out. For example:

```
cWebState = getWebState().
```

2.) Sets the Web object's Web state to state–aware and its timeout. The number of minutes to remain state–aware. For example:

```
lReturn = setWebState(5.0).
```

WebTimeout

Returns a state–aware Web object's timeout in minutes. For example:

```
dWebTimeout = getWebTimeout().
```

WebTimeRemaining

Returns the time remaining for a state–aware Web object. For example:

```
dWebTimeout = getWebTimeRemaining().
```

WebToHdlr

1.) Returns the name of the Web object/procedure to run when the state–aware Web object that is currently running times out. The name of the object/procedure to run. For example:

```
cWebToHdlr = getWebToHdlr().
```

2.) Sets the name of the Web object/procedure to run when the state–aware Web object that is currently running times out. The Web-object name.Web object must be on the Agent's PROPATH. For example:

```
lReturn = setWebToHdlr("mytohdlr.w").
```

Width

Returns the width of the object.

Window

Returns the handle of the toolbar's parent Window.

WindowFrameHandle

The {get} pseudo-function returns the value of WindowFrameHandle. This value is the handle of the optional Frame widget of a Window container.

- This property ONLY identifies the frame of a window container. It must not be confused with the important ContainerHandle, which is the widget handle of the container in all object. In most cases the ContainerHandle will be the Frame handle also for a SmartContainer.
- Even if Window containers do not need a frame they often have one and in that case we must include it when, for example, resizing, widget-tree logic, etc.

WindowTitleField

The {get} pseudo-function returns the value of WindowTitleField.

WindowTitleViewer

The {get} pseudo-function returns the value of WindowTitleViewer.

WordIndexedFields

1.) A comma-separated list of RowObject fields that are mapped to database fields that have a word indexed.

2.) A comma-separated list of word indexed fields.

Index

A

action.p
- actionAccelerator 60
- actionAccessType 60
- actionCaption 61
- actionCategory 61
- actionChildren 63
- actionControlType 63
- actionCreateEvent 64
- actionDescription 64
- actionDisabled 64
- actionEnableRule 65
- actionEnableRulex 65
- actionGroups 65
- actionHideRule 66
- actionHideRuleX 66
- actionImage 66
- actionImageAlternate 67
- actionImageAlternateRule 67
- actionInitCode 67
- actionIsMenu 68
- actionIsParent 68
- actionLabel 68
- actionLink 69
- actionLogicalObjectName 69
- actionName 70
- actionOnChoose 70
- actionParameter 70
- actionParent 71
- actionPhysicalObjectName 71
- actionRefresh 72
- actionRunAttribute 73
- actionSecondImage 73
- actionSecuredToken 73
- actionSubstituteProperty 74
- actionTooltip 74
- actionType 75
- assignActionAccelerator 95
- assignActionAccessType 95
- assignActionAlternateImageRule 96
- assignActionCaption 96
- assignActionDescription 97
- assignActionDisableRule 97
- assignActionEnableRule 98
- assignActionHideRule 98
- assignActionImage 99
- assignActionLabel 99
- assignActionName 100
- assignActionOrder 100
- assignActionParent 101
- assignActionSecondImage 101
- assignActionTooltip 102
- assignColumn 103
- bufferHandle 122
- canFindAction 128
- canFindCategory 129
- categoryLink 131
- clearActionCache 134
- clearCache 134
- columnStringValue 155
- defineAction 205
- destroyObject 212

(continued)

action.p *(continued)*
displayActions 224
errorMessage 244
findAction 257
findCategory 258
initAction 269
InitializeObject 270
initializeObject 270
loadActions 297
loadBands 298
loadCategories 298
loadObjectBands 299
loadToolbarBands 301
normalizeActionData 318
validateBuffer 431

ActionEvent property 1

ActionGroups property 1

admweb.p
destroy 464
destroyObject 465
timingOut 485

AllFieldNames property 1

ApplyActionOnExit property 1

ApplyExitOnAction property 1

appserver.p
bindServer 118
destroyObject 212
destroyServerObject 217
disconnectObject 224
initializeServerObject 280
restartServerObject 363
runServerObject 373
runServerProcedure 374
startServerObject 398, 401
unbindServer 417

AppService property 2

ASBound property 2

ASDivision property 2

ASHandle property 3

ASHasStarted property 2

ASInfo property 2

ASInitializeOnRun property 2

AssignList property 3

ASUseConnection property 3

ASUsePrompt property 3

AsynchronousSDO property 3

Audience xxvii

AutoCommit property 3

AutoSort property 3

AvailMenuActions property 3

AvailToolbarActions property 4

AvailToolbarBands property 4

B

b2b.p
callOutParams 124
characterValue 133
createSchemaAttributes 194
createSchemaChildren 195
createSchemaPath 195
dataSource 203
defineMapping 206
destroyObject 213
endDocument 241
endElement 241
findDataRow 258
InitializeObject 271
initializeObject 271
loadProducerSchema 300
loadSchema 300
mapNode 302
NotFoundError 319
numParameters 320
processMappings 335
processMessages 336
(continued)

b2b.p *(continued)*
produceAttributes 337
produceChildren 337
produceDocument 337
receiveHandler 341
rowNotFoundError 369
schemaField 375
sendHandler 378
sendMessage 379
startDataRow 400
startElement 400
storeNodeValue 403
storeParameterNode 403
storeParameterValue 404
targetProcedure 413

BaseQueryString property 4

BlockDataAvailable property 4

Bold typeface
as typographical convention xxix

BoxRectangle property 4

BoxRectangle2 property 4

BrowseFieldDataTypes property 4

BrowseFieldFormats property 4

BrowseFields property 4

BrowseHandle property 4

browser.i
dataAvailable 199

browser.p
addRecord 84
applyCellEntry 91
applyEntry 92
assignMaxGuess 112
calcWidth 123
cancelNew 124
cancelRecord 126
colValues 164
copyRecord 177
defaultAction 204
deleteComplete 206
deleteRecord 209
destroyObject 213
disableFields 221
displayFields 225
enableFields 237
fetchDataSet 249
filterActive 256
hasActiveAudit 263
hasActiveComments 264
initializeObject 271
launchFolderWindow 292
offEnd 321
offHome 321
refreshBrowse 343
resetRecord 355
resizeBrowse 357
resizeObject 357
rowDisplay 368
rowVisible 372
searchTrigger 376
startSearch 398
stripCalcs 405
toolbar 413
updateRecord 423, 424
updateState 425
updateTitle 429
viewObject 434

BrowseTitle property 5

BuildSequence property 5

ButtonCount property 5

C

CalculatedColumns property 5

CalcWidth property 5

CallerObject property 5

CallerProcedure property 6

CallerWindow property 6

CancelBrowseOnExit property 6

CascadeOnBrowse property 6

ChangedEvent property 6

CheckCurrentChanged property 6

CheckLastOnOpen property 6

ChildDataKey property 6

ClientID property 7

ClientProxyHandle property 7

ClientRect property 7

Col property 7

ColumnFormat property 7

ColumnLabels property 7

combo.p
- anyKey 89
- createLabel 185
- destroyCombo 212
- destroyObject 213
- disable_UI 218
- disableField 220
- displayCombo 224
- enableField 236
- endMove 242
- enterCombo 243
- hideObject 265
- initializeCombo 270
- insertExpression 283
- leaveCombo 292
- newQueryString 311
- newWhereClause 316
- refreshChildDependancies 344
- refreshCombo 344
- resizeObject 358
- returnParentFieldValues 365
- showQuery 396
- valueChanged 432
- viewObject 435
- whereClauseBuffer 439

ComboDelimiter property 7

ComboFlag property 7

ComboFlagValue property 7

ComboHandle property 7

ComboQuery property 8

ComboSort property 8

CommitSource property 8

CommitSourceEvents property 8

CommitTarget property 8

CommitTargetEvents property 8

consumer.p
- assignUnsubscribe 116
- createConsumers 182
- defineDestination 205
- destroyObject 214
- errorHandler 244
- initializeObject 272
- messageHandler 304
- processDestinations 334
- startWaitFor 402
- stopHandler 402

ConsumerSchema property 8

ContainedDataColumns property 8

ContainedDataObjects property 9

ContainerHandle property 9

ContainerHidden property 9

ContainerMode property 9

ContainerSource property 9

ContainerSourceEvents property 9

ContainerTarget property 9

ContainerTargetEvents property 9

ContainerType property 9

containr.p
assignPageProperty 113
cancelObject 125
changePage 132
confirmCancel 167
confirmExit 171
confirmOk 172
constructObject 174
createObjects 192
deletePage 208
destroyObject 214
disablePagesInFolder 223
enablePagesInFolder 240
hidePage 267
initializeObject 272
initializeVisualContainer 280
initPages 281
isUpdateActive 289
notifyPage 319
okObject 321
pageNTargets 326
passThrough 327
removePageNTarget 349
resizeWindow 362
selectPage 377
targetPage 412
toolbar 414
updateActive 420
viewObject 435
viewPage 437

ContextandDestroy property 9

ContextAndInitialize property 10

CreateHandles property 10

CtrlFrameHandle property 10

CurrentDescValue property 10

CurrentKeyValue property 10

CurrentMessage property 10

CurrentMessageId property 11

CurrentPage property 11

CurrentRowids property 11

CurrentRowModified property 11

D

data.i
canFindModRow 129
Commit 166
doBuildUpd 230
doCreateUpdate 231
doEmptyModTable 231
doEmptyTempTable 232
doReturnUpd 232
doUndoDelete 233
doUndoRow 233
doUndoTrans 233
doUndoUpdate 234
prepareErrorsForReturn 329
pushTableAndValidate 338
serverCommit 383
serverSendRows 391
undoClientUpdate 419

data.p
addRow 87
batchServices 117
cancelRow 128
canNavigate 129
clientSendRows 135
closeQuery 136
columnProps 152
colValues 164
commitTransaction 166
copyColumns 176
copyRow 179
createRow 194
dataAvailable 199
deleteRow 210
describeSchema 211
destroyObject 214
destroyServerObject 217
endClientDataRequest 240
fetchBatch 247
fetchFirst 250
fetchLast 251
fetchNext 252
(continued)

data.p *(continued)*
fetchPrev 253
fetchRow 254
fetchRowIdent 255
findRow 259
findRowObjUpd 259
findRowWhere 260
firstRowIds 261
genContextList 263
hasActiveAudit 263
hasActiveComments 264
hasForeignKeyChanged 264
hasOneToOneTarget 265
initializeObject 272
initializeServerObject 280
isUpdateActive 289
isUpdatePending 290
newRowObject 315
openDataQuery 324
openQuery 325
prepareQuery 331
printToCrystal 332
refreshRow 345
repositionRowObject 353
restartServerObject 363
retrieveFilter 364
rowAvailable 367
rowObjectState 370
rowValues 372
saveContextAndDestroy 375
sendRows 382
serverFetchRowObjUpdTable 390
startServerObject 399, 401
submitCommit 406
submitForeignKey 406
submitRow 408
submitValidation 409
synchronizeProperties 410
tableOut 411
transferToExcel 416
undoTransaction 419
updateAddQueryWhere 421
updateManualForeignFields 422
updateQueryPosition 423
updateRow 424
updateState 426

DataColumns property 12

dataextcols.p
assignColumnColumnLabel 103
assignColumnFormat 104
assignColumnHelp 105
assignColumnLabel 106
assignColumnPrivateData 107
assignColumnValExp 108
assignColumnValMsg 109
columnColumnLabel 138
columnDataType 139
columnExtent 141
columnFormat 142
columnHandle 144
columnHelp 144
columnInitial 145
columnLabel 146
columnModified 148
columnPrivateData 150
columnQuerySelection 152
columnReadOnly 154
columnStringValue 156
columnTable 157
columnValExp 159
columnValMsg 160
columnValue 161
columnWidth 162

DataFieldDefs property 12

DataHandle property 12

DataIsFetched property 12

DataLinksEnabled property 12

datalogic.i
beginTransactionValidate 118
endTransactionValidate 242
initializeLogicObject 270
postTransactionValidate 329
preTransactionValidate 332
rowObjectValidate 371

DataLogicObject property 13

datamessage.p
serverCommit 384
serverCommitTransaction 385
serverFetchContainedData 387
serverFetchContainedRows 388
serverSendRows 392

DataModified property 13

DataObject property 13

DataObjectHandle property 13

DataObjectNames property 13

DataObjectOrdering property 14

DataQueryBrowsed property 14

DataQueryString property 14

DataSignature property 14

DataSource property 14

DataSourceEvents property 14

DataSourceFilter property 15

DataSourceNames property 15

DataTarget property 15

DataTargetEvents property 15

DataValue property 15

datavis.i
displayObjects 229

datavis.p
addRecord 85
cancelObject 125
cancelRecord 126
canNavigate 130
collectChanges 137
confirmCancel 168
confirmCommit 168
confirmContinue 169
confirmDelete 170
confirmExit 171
confirmOk 172
confirmUndo 173
copyRecord 178
dataAvailable 200
deleteRecord 209
disableFields 221
displayRecord 229
enableFields 237
initializeObject 273
isUpdateActive 290
okObject 322
okToContinueProcedure 322
queryPosition 340
resetRecord 355
showDataMessages 394
showDataMessagesProcedure 395
updateMode 422
updateRecord 424
updateState 426
updateTitle 430
validateFields 431

DBAware property 15

DBNames property 15

DC property 15

DefaultCharWidth property 16

DefaultEditorLines property 16

DefaultLayout property 16

DefaultLogical property 16

DefaultWidth property 16

DefineAnyKeyTrigger property 16

DescSubstitute property 16

design.p
destroyObject 214
linkHandles 293

DesignDataObject property 16

DesignTimeHideMenu property 16

Destination property 17

DestinationList property 17

Destinations property 17

DestroyStateless property 17

DeviceCaps property 17

DirectionList property 17

DisabledActions property 17

DisabledAddModeTabs property 17

DisableOnInit property 17

DisconnectAppServer property 17

DisplayDataType property 17

DisplayedField property 18

DisplayedFields property 18

DisplayedTables property 18

DisplayField property 18

DisplayFieldsSecurity property 18

DisplayFormat property 18

DisplayValue property 18

DocumentElement property 18

DocumentHandle property 18

DocumentInitialized property 18

Domain property 19

Down property 19

DTDPublicId property 19

DTDPublicIdList property 19

DTDSystemID property 19

DTDSystemIdList property 19

DynamicObject property 19

DynamicSDOProcedure property 19

E

EdgePixels property 19

Editable property 20

EnabledFields property 20

EnabledHandles property 20

EnabledObjFlds property 20

EnabledObjHdls property 20

EnabledTables property 20

EnableField property 20

EnableOnAdd property 20

Error messages
displaying descriptions xxxvii

Example procedures xxxiv

ExitBrowseOnAction property 20

ExpandOnAdd property 20

ExternalRefList property 21

ExternalTableList property 22

ExternalTables property 22

ExternalWhereList property 23

F

field.p
initializeObject 273
resizeObject 359

FieldColumn property 23

FieldEnabled property 23

FieldFormats property 23

FieldHandles property 23

FieldHelpIds property 24

FieldHidden property 24

FieldLabel property 24

FieldLabels property 24

FieldName property 24

FieldOperatorStyles property 24

FieldsEnabled property 24

FieldToolTip property 24

FieldToolTips property 24

FieldWidths property 24

FillBatchOnRepos property 25

filter.p
- applyFilter 94
- assignColumnFormat 104
- assignColumnHelpId 105
- assignColumnLabel 106
- assignColumnOperatorStyle 107
- assignColumnTooltip 108
- assignColumnWidth 109
- bBlankLogical 418
- blankField 118
- blankFields 119
- blankFillIn 119
- columnDataType 139
- columnFIlterTarget 142
- columnFilterTarget 142
- columnFormat 143
- columnHelpId 145
- columnLabel 147
- columnLabelDefault 147
- columnOperatorStyle 150
- columnStyleDefault 157
- columnTooltip 159
- columnWidth 162
- columnWidthDefault 163
- createField 183
- createLabel 185
- createOperator 193
- dataAvailable 201
- dataValue 203
- deleteObjects 207
- disableFields 221
- enableFields 238
- fieldLookup 255
- fieldProperty 256
- initializeObject 274
- insertFieldProperty 285
- removeSpace 351
- resetFields 354
- showDataMessages 394
- unBlankFillin 418
- unBlankLogical 418

FilterActive property 25

FilterAvailable property 25

FilterSource property 25

FilterTarget property 25

FilterTargetEvents property 25

FilterWindow property 25

FirstResultRow property 25

FirstRowNum property 26

FlagValue property 26

FlatButtons property 26

FolderWindowToLaunch property 26

ForeignFieldList property 26

ForeignFields property 26

ForeignValues property 26

Format property 26

FrameMinHeightChars property 27

FrameMinWidthChars property 27

FullRowSelect property 27

FuncLibHandle property 27

G

genContextList property 1

GroupAssignSource property 27

GroupAssignSourceEvents property 27

GroupAssignTarget property 27

GroupAssignTargetEvents property 27

H

Height property 28

Help
  Progress messages xxxvii

HelpId property 28

HiddenActions property 28

HiddenMenuBands property 28

HiddenToolbarBands property 28

HideOnInit property 28

HideSelection property 28

html-map.p
  assignFields 454
  bufferHandle 456
  columnHTMLName 458
  columnTable 460
  constructObject 462
  dataAvailable 462
  deleteOffsets 463
  dispatchUtilityProc 466
  displayFields 467
  enableFields 468
  findRecords 473
  htmAssociate 474
  initializeObject 476
  inputFields 476
  outputFields 479
  readOffsets 483

I

ILComHandle property 28

ImageHeight property 28

ImagePath property 29

ImageWidth property 29

Indentation property 29

IndexInformation property 29

InitialPageList property 29

InMessageSource property 29

InMessageTarget property 29

InnerLines property 29

InstanceProperties property 30

InternalEntries property 30

InternalRefList property 30

Italic typeface
  as typographical convention xxix

J

JMSpartition property 30

JMSpassword property 30

JMSsession property 30

JMSuser property 30

K

KeyDataType property 30

KeyField property 30

KeyFields property 31

KeyFieldValue property 31

KeyFormat property 31

Keystrokes xxix

L

Label property 31

LabelEdit property 31

LabelHandle property 31

LastResultRow property 31

LastRowNum property 31

LayoutOptions property 32

LayoutVariable property 32

LineStyle property 32

LinkColumns property 32

LinkedFieldDataTypes property 32

LinkedFieldFormats property 32

ListItemPairs property 32

LoadedByRouter property 32

LogFile property 32

logic.i
  clearLogicRows 134
  serverCommit 384

LogicalObjectName property 33

LogicalVersion property 33

LogicBuffer property 33

lookup.p
  anyKey 89
  createLabel 186
  destroyLookup 212
  destroyObject 215
  disable_UI 218
  disableButton 219
  disableField 220
  displayLookup 228
  enableButton 236
  enableField 236
  endMove 242
  enterLookup 243
  hideObject 265
  initializeBrowse 269
  initializeLookup 270
  insertExpression 284
  leaveLookup 292
  newQueryString 312
  newWhereClause 317
  resizeObject 359
  returnParentFieldValues 365
  rowSelected 371
  translateBrowseColumns 416
  valueChanged 432
  viewObject 436
  whereClauseBuffer 439

LookupFilterValue property 33

LookupHandle property 33

LookupImage property 33

LookupQuery property 33

M

MaintenanceObject property 34

MaintenanceSDO property 34

Manual
  syntax notation xxx

Manual, organization of xxvii

ManualAddQueryWhere property 34

ManualAssignQuerySelection property 34

ManualSetQuerySort property 34

MapNameProducer property property 34

MapObjectProducer property property 34

MapTypeProducer property property 34

MarginPixels property 34

MasterDataObject property 35

MaxWidth property 35

Menu property 35

MenuMergeOrder property 35

Messages
  displaying descriptions xxxvii

MessageType property 35

messaging.p
  destroyObject 215
  errorHandler 244
  initializeObject 274

MinHeight property 35

MinWidth property 35

Monospaced typeface
  as typographical convention xxix

msghandler.p
  sendMessage 379

MultiInstanceActivated property 35

MultiInstanceSupported property 35

N

NameList property property 35

NameSpaceHandle property 35

NavigationSource property 36

NavigationSourceEvents property 36

NavigationTarget property 36

NavigationTargetEvents property 36

NavigationTargetName property 36

NewNode property 36

NewRecord property 36

NewRow property 36

NextNodeKey property 37

NumDown property 37

NumRows property 37

O

ObjectEnabled property 37

ObjectHidden property 37

ObjectInitialized property 37

ObjectLayout property 37

ObjectMapping property 38

ObjectMode property 38

ObjectName property 38

ObjectPage property 38

ObjectParent property 38

ObjectType property 38

ObjectVersion property 38

OLEDrag property 38

OLEDrop property 39

OpenOnInit property 39

OpenQuery property 39

Operator property 39

OperatorLongValues property 39

OperatorStyle property 39

OperatorViewAs property 39

Optional property 40

OptionalBlank property 40

OptionalString property 40

OutMessageSource property 40

OutMessageTarget property 40

P

PageNTarget property 40

PageSource property 40

panel.p
- activeTarget 76
- checkRule 133
- countButtons 180
- disableActions 218
- enableActions 235
- enableObject 239
- hasActiveGaTarget 264
- initializeObject 275
- linkState 297
- loadPanel 300
- onChoose 323
- queryPosition 340
- resetCommit 353
- resetNavigation 354
- resetTableio 356
- resetTargetActions 356
- resizeObject 360
- sensitizeActions 381
- targetActions 412
- updateState 427
- viewHideActions 433

PanelFrame property 40

PanelLabel property 40

PanelState property 41

PanelType property 41

Parent property 41

ParentDataKey property 41

ParentField property 41

ParentFilterQuery property 41

PassThroughLinks property 41

Persistency property 41

PhysicalObjectName property 41

PhysicalVersion property 41

PingInterval property 41

PrimarySdoTarget property 41

Priority property 41

Procedures
- examples of xxxiv

processCDataSection (xml.p 333

producer.p
- destroyObject 215
- initializeObject 275
- replyHandler 351
- sendMessage 379

PromptLogin property 42

Property property 42

PropertyDialog property 43

Q

query.i
- deleteRecordStatic 210
- initProps 282

query.p
- addQueryWhere 83
- assignDBRow 111
- assignQuerySelection 114
- bufferCompareDBToRO 120
- bufferCopyDBToRO 121
- bufferWhereClause 122
- *(continued)*

query.p *(continued)*
closeQuery 136
columnDataType 140
columnDbColumn 141
columnQuerySelection 153
columnTable 158
columnValMsg 160
colValues 165
compareDBRow 167
confirmCommit 168
confirmContinue 169
confirmUndo 173
dataAvailable 201
dbColumnDataName 204
dbColumnHandle 204
excludeColumns 245
fetchDBRowForUpdate 249
fetchFirst 250
fetchLast 251
fetchNext 252
fetchPrev 253
filterContainerHandler 257
firstBufferName 260
firstRowIds 261
firstRowsIds 261
indexInformation 268
initializeObject 276
insertExpression 284
newQueryString 313
newQueryValidate 314
newQueryWhere 315
newWhereClause 317
openQuery 325
prepareQuery 331
refetchDBRow 343
refreshRowident 345
releaseDBRow 346
removeForeignKey 347
removeQuerySelection 349
resolveBuffer 362
rowidWhere 368
rowidWhereCols 369
startFilter 401
transferDBRow 415
whereClauseBuffer 440

QueryColumns property 43

QueryContainer property 43

QueryContext property 43

QueryHandle property 43

QueryObject property 43

QueryOpen property 43

QueryPosition property 44

QueryRowIdent property 44

QueryRowObject property 44

QuerySort property 44

QueryString property 44

QueryTables property 44

QueryWhere property 44

R

RebuildOnRepos property 45

RecordState property 45

ReEnableDataLinks property 45

RefreshList property 45

ReplyReqList property 45

ReplyRequired property 45

ReplySelector property 45

ReplySelectorList property 45

RepositionDataSource property 45

ResizeHorizontal property 46

ResizeVertical property 46

RootNodeParentKey property 46

router.p
createDocument 182
initializeObject 276
internalSchemaFile 287
obtainInMsgTarget 320
processFileRefs 335
routeBytesMessage 366
routeDocument 366
routeMessage 367
startB2BObject 399

RouterSource property 46

RouterTarget property 46

Row property 46

RowIdent property 46

RowObject property 47

RowObjectState property 47

RowObjectTable property 47

RowObjUpd property 47

RowObjUpdTable property 47

RowsToBatch property 47

RunAttribute property 47

RunDOOptions property 48

RunMultiple property 48

S

SavedContainerMode property 48

SavedScreenValue property 48

SaveSource property 48

sbo.i
initDataObjectOrdering 269
serverCommitTransaction 385

sbo.p
addDataTarget 77
addNavigationSource 81
addQueryWhere 84
addRow 88
appendContainedObjects 91
assignContainedProperties 110
assignCurrentMappedObject 110
assignMaxGuess 112
assignQuerySelection 115
cancelNew 124
cancelRow 128
canNavigate 130
columnColumnLabel 138
columnDataType 140
columnDbColumn 141
columnExtent 142
columnFormat 143
columnHelp 144
columnInitial 146
columnLabel 147
columnMandatory 148
columnModified 149
columnObjectHandle 149
columnPrivateData 151
columnProperty 151
columnQuerySelection 154
columnReadOnly 155
columnStringValue 156
columnTable 158
columnValExp 159
columnValMsg 160
columnValue 161
columnWidth 163
colValues 165
commitTransaction 166
confirmContinue 170
containedProperties 176
copyRow 179
currentMappedObject 198
dataAvailable 202
dataObjectHandle 203
deleteComplete 206
deleteRow 210
destroyServerObject 218
endClientDataRequest 240
fetchBatch 247
fetchContainedData 248
(continued)

sbo.p *(continued)*
fetchContainedRows 248
fetchDOProperties 249
fetchFirst 250
fetchLast 251
fetchNext 253
fetchPrev 253
initializeObject 276
initializeServerObject 280
isUpdatePending 291
newDataObjectRow 310
openQuery 325
postCreateObjects 329
prepareErrorsForReturn 330
prepareQueriesForFetch 330
queryPosition 341
refreshBrowse 343
registerLinkedObjects 344
registerObject 346
removeQuerySelection 350
repositionRowObject 353
resetQuery 354
restartServerObject 364
serverContainedSendRows 386
serverFetchContainedData 389
serverFetchContainedRows 389
serverFetchDOProperties 390
startServerObject 399, 401
submitRow 408
undoTransaction 419
updateState 427

SchemaHandle property 48

SchemaList property 48

SchemaManager property 48

Scroll property 49

ScrollRemote property 49

SDFFileName property 49

SDFTemplate property 49

SdoForeignFields property 49

SearchField property 49

Secured property 49

SecuredTokens property 49

select.p
anyKey 90
browseHandler 120
buildList 123
createLabel 186
dataAvailable 202
destroyObject 215
destroySelection 217
disable_UI 218, 219
disableButton 219
disableField 220
enableButton 236
enableField 237
endMove 242
enterSelect 243
formattedValue 262
hideObject 265
initializeBrowse 269
initializeObject 277
initializeSelection 279
leaveSelect 293
queryOpened 339
refreshObject 345
repositionDataSource 351
resizeObject 360
rowSelected 371
valueChanged 432
viewObject 436

SelectedNode property 49

Selectors property 49

ServerFileName property 50

ServerOperatingMode property 50

ServerSubmitValidation property 50

ShowBorder property 50

ShowCheckBoxes property 50

ShowRootLines property 50

ShutDownDest property 50

SingleSel property 51

smart.i
start-super-proc 399

smart.p
addLink 78
addMessage 80
adjustTabOrder 88
anyMessage 90
applyEntry 93
assignLinkProperty 111
changeCursor 131
createControls 182
destroyObject 216
displayLinks 228
editInstanceProperties 234
exitObject 246
fetchMessages 252
fixQueryString 262
hideObject 266
initializeObject 277
instancePropertyList 286
isObjQuoted 288
linkHandles 294
linkProperty 296
mappedEntry 303
messageNumber 305
modifyListProperty 306
modifyUserLinks 308
oneObjectLinks 323
propertyType 338
removeAllLinks 347
removeLink 348
repositionObject 352
returnFocus 364
reviewMessages 365
showMessage 395
showMessageProcedure 396
Signature 397
toggleData 413
viewObject 436

Sort property 51

StartBrowseKeys property 51

StaticPrefix property 51

SubModules property 51

Subscriptions property 51

SupportedLinks property 51

SupportedMessageTypes property 51

Syntax notation xxx

T

TableIOButtons property 51

TableIOSource property 51

TableIOSourceEvents property 51

TableioTarget property 52

TableioTargetEvents property 52

TableioType property 52

Tables property 53

TargetNameSpace property 53

TargetProcedure property 53

TimeToLive property 53

Toolbar property 53

toolbar.p
actionCanRun 60
actionCaption 61
actionCategoryIsHidden 62
actionChecked 62
actionLabel 69
actionPublishCreate 72
actionTarget 74
actionTooltip 75
adjustActions 88
bandActions 116
bandSubmenuLabel 116
bandSubmenus 117
buildAllMenus 123
buildMenu 123
categoryActions 131
clearCache 134
clearToolbarCache 135
constructMenu 173
(continued)

toolbar.p *(continued)*
constructMenuBand 174
constructObjectMenus 175
constructToolbar 175
create3DRule 180
createButton 181
createMenuAction 187
createMenuBar 187
createMenuItem 188
createMenuTable 188
createMenuTable2 189
createMenuToggle 190
createRule 194
createSubMenu 196
createToolBar 197
createToolbar 198
createToolbarAction 197
createToolbarBorder 198
deleteMenu 207
deleteToolbar 211
destroyObject 216
filterState 257
hasActiveGaTarget 264
hideObject 267
imageName 268
initializeObject 278
insertMenu 286
linkState 297
loadToolbar 300
menuHandle 303
modifyDisabled Actions 306
modifyDisabledActions 306
movebuttons 309
moveMenu 309
moveMenuChildren 310
onChoose 323
onMenuDrop 324
onValueChanged 324
queryPosition 341
repositionObject 352
resetTargetActions 356
resetToolbar 357
resizeObject 361
rowObjectState 370
runInfo 373
sensitizeActions 383
storePendingSensitivity 405
substituteActionText 409
supportedObjects 410
targetActions 412
updateActive 420
updateCategoryLists 421
updateState 428
updateStates 429
viewHideActions 434
viewObject 437
windowDropDownList 440
windowEnabled 441
windowFocus 441
windowListMenu 442

ToolbarAutoSize property 53

ToolbarBands property 54

ToolbarDrawDirection property 54

ToolbarHeightPxl property 54

ToolbarInitialState property 54

ToolbarParentMenu property 54

ToolbarSource property 54

ToolbarSourceEvents property 54

ToolbarTarget property 54

ToolbarTargetEvents property 54

ToolbarWidthPxl property 54

ToolMarginPxl property 54

ToolTip property 55

TopOnly property 55

TranslatableProperties property 55

TreeDataTable property 55

TreeStyle property 55

treeview.i
ReleaseDC 347
SendMessageA 380
ShellExecuteA 393

treeview.p
addNode 82
deleteNode 207
deleteTree 211
disableObject 223
emptyTree 235
enableObject 239
isNodeExpanded 288
loadImage 299
populateTree 327
repositionObject 352
resizeObject 361
selectFirstNode 376
selectNode 376
showTVError 397
updateTree 430

tvcontroller.p
initializeObject 278
putParam 339
showTVCError 397
updateState 428

TVControllerSource property 55

TVControllerTarget property 55

TVControllerTargetEvents property 56

TypeName property 56

Typographical conventions xxix

U

UIBMode property 56

UpdatableColumns property 56

UpdateActive property 56

UpdateFromSource property 56

UpdateMode property 56

UpdateSource property 57

UpdateTarget property 57

UpdateTargetNames property 57

UseBegins property 57

UseContains property 57

UseDBQualifier property 57

UseDTD property 57

UsePairedList property 58

UseRepository property 58

UserNameA property 58

UserProperty property 58

V

ValidateOnLoad property 59

ValidKey property 59

ViewAs property 59

ViewAsFields property 59

viewer.p
addRecord 86
cancelRecord 127
copyRecord 178
disableFields 222
displayFields 225
enableFields 238
initializeObject 278
toolbar 415
updateState 429
valueChanged 433
viewRecord 438

ViewerLinkedFields property 59

ViewerLinkedWidgets property 59

VisibleRowids property 59

VisibleRowReset property 59

visual.p
applyLayout 94
disableObject 223
enableObject 239
initializeObject 279
processAction 333

VisualBlank property 59

W

WaitForObject property 60

Waiting property 60

wbdata.p
assignFields 455
columnValMsg 461
deleteBuffer 463
deleteRow 464
exclusiveLockBuffer 468
lockRow 478
processWebRequest 480
setAddMode 484
validateColumns 487

wbrep.p
HTMLAlert 474
HTMLSetFocus 475
joinExternalTables 477
joinForeignFields 478

wbtable.p
addColumnLink 446
addTDModifier 450
assignTDModifier 456
columnTDModifier 461
fetchLast 470
fetchNext 471
fetchPrev 472
HTMLColumn 475
HTMLTable 476
pageBackward 480
processWebRequest 481

webrep.p
addContextFields 449
addSearchCriteria 449
anyMessage 450
assignColumnFormat 451
assignColumnHelp 451
assignColumnLabel 452
assignColumnWidth 452
assignExtentAttribute 453
bufferHandle 457
columnDataType 457
columnFormat 457
columnHandle 458
columnHelp 458
columnHTMLName 459
columnLabel 459
columnReadOnly 460
columnStringValue 460
destroyDataObject 464
destroyObject 465
disconnectObject 465
extentAttribute 469
extentValue 469
fetchCurrent 470
fetchFirst 470
fetchLast 471
fetchNext 471
fetchPrev 472
fieldExpression 472
openQuery 479
processWebRequest 482
removeEntry 483
reOpenQuery 483
rowidExpression 484
showDataMessages 484
startDataObject 485
urlJoinParams 486
urlLink 486
validateColumnValue 488

WebTimeout property 60

WebTimeRemaining property 60

WebToHdlr property 61

Width property 61

Window property 61

WindowFrameHandle property 61

WindowTitleField property 61

WindowTitleViewer property 61

WordIndexedFields property 61

X

xml.p
- assignAttribute 102
- assignNodeValue 113
- createAttribute 180
- createDocument 183
- createElement 183
- createNode 191
- createText 196
- deleteDocument 207
- destroyObject 217
- nodeHandle 318
- nodeType 318
- ownerElement 325
- parentNode 326
- processCDataSection 333
- processComment 333
- processDocument 334
- processElement 335
- processRoot 336
- processText 336
- receiveHandler 342
- receiveReplyHandler 342
- sendHandler 378
- sendReplyHandler 381