



Progress Dynamics. ADM2 API Reference

© 2005 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.

A [Stylized], Allegrix, Allegrix & Design, Business Empowerment, eXcelon, ObjectStore, PeerDirect, Progress, Progress Dynamics, Powered by Progress, Empowerment Center, Progress Empowerment Center, Progress Empowerment Program, Progress Fast Track, Progress OpenEdge, Progress Profiles, Partners in Progress, Partners en Progress, Progress en Partners, Progress in Progress, P.I.P., Progress Results, Progress Software Developers Network, ProVision, ProCare, ProtoSpeed, SmartBeans, SpeedScript, Technical Empowerment, and WebSpeed are registered trademarks of Progress Software Corporation or one of its subsidiaries or affiliates in the U.S. and/or other countries. AccelEvent, A Data Center of Your Very Own, AppsAlive, AppServer, ASPen, ASP-in-a-Box, BusinessEdge, Cache-Forward, Fathom, Future Proof, IntelliStream, ObjectCache, ObjectStore Event Engine, ObjectStore RFID Accelerator, ObjectStore Trading Accelerator, OpenEdge, POSSE, POSSENET, ProDataSet, Progress Business Empowerment, Progress for Partners, PSE Pro, PS Select, SectorAlliance, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, WebClient, and Who Makes Progress are trademarks or service marks of Progress Software Corporation or one of its subsidiaries or affiliates 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 service marks contained herein are the property of their respective owners.

February 2005



Product Code: 4482
Item Number: 103618;V2.1B

Contents

Preface	xxxix
Purpose	xxxix
Audience	xxxix
Organization of this manual	xxxix
How to use this manual	xli
Typographical conventions	xli
Syntax notation	xlvi
Progress messages	xlvi
1. ADM2 SmartObject API Reference	1-1
Paths to source files	1-2
Source-file types	1-2
ADM class hierarchy	1-4
SmartObjects, templates, and super-procedure hierarchies	1-5
SmartObjects and their files	1-8
SmartObject	1-8
SmartContainer	1-8
SmartBusinessObject	1-9
SmartB2BObject	1-10
SmartCombo	1-12
SmartConsumer	1-13
SmartDataBrowser	1-13
SmartDataField	1-14
SmartDataObject	1-14
SmartDataViewer	1-16
SmartDialog	1-17
SmartFilter	1-17
SmartFolder	1-18
SmartFrame	1-19

SmartLookup	1-20
SmartPanel.	1-20
SmartProducer and SmartConsumer	1-21
Messaging class	1-23
SmartRouter	1-23
SmartSelect	1-24
SmartSender and SmartReceiver	1-24
SmartToolbar	1-25
SmartWindow	1-26
Reading and writing object properties	1-27
<i>getpropname</i> and <i>setpropname</i> functions	1-27
{get} and {set} pseudo-functions	1-27
The need for two different syntaxes	1-28
2. SmartObjects and Their Methods and Properties	2-1
Base methods for SmartObjects	2-2
addLink	2-2
addMessage.	2-4
adjustTabOrder	2-5
anyMessage.	2-6
applyEntry	2-6
assignLinkProperty	2-7
changeCursor.	2-8
createControls	2-8
destroyObject	2-9
displayLinks	2-10
editInstanceProperties	2-10
exitObject	2-11
fetchMessages	2-12
fixQueryString	2-13
initializeObject	2-13
instanceOf	2-14
instancePropertyList	2-15
hideObject	2-16
linkHandles.	2-17
linkProperty	2-18
linkStateHandler.	2-19
mappedEntry	2-20
messageNumber	2-21
modifyListProperty	2-22
modifyUserLinks.	2-23
oneObjectLinks.	2-24
propertyType	2-24
removeAllLinks.	2-25

removeLink	2-25
repositionObject	2-26
returnFocus	2-27
reviewMessages	2-27
showMessage	2-28
showMessageProcedure.	2-28
Signature.	2-29
start-super-proc.	2-29
viewObject.	2-29
AppServer methods for SmartObjects	2-30
bindServer.	2-30
destroyObject	2-31
destroyServerObject	2-31
disconnectObject.	2-31
initializeServerObject	2-31
runServerObject	2-32
runServerProcedure	2-32
unbindServer.	2-32
SmartObject properties	2-33
AppService	2-34
ASBound	2-34
AsDivision	2-34
ASHandle	2-35
ASHasStarted	2-35
ASInitializeOnRun	2-35
ChildDataKey	2-35
ContainerHandle	2-36
ContainerHidden	2-36
ContainerSource	2-36
ContainerSourceEvents	2-36
ContainerType.	2-37
DataLinksEnabled	2-37
DataSource	2-37
DataSourceEvents	2-37
DataSourceNames	2-37
DataTarget	2-38
DataTargetEvents	2-38
DBAware	2-38
inactiveLinks	2-38
InstanceProperties	2-38
LogicalObjectName.	2-39
LogicalVersion.	2-39
ObjectHidden	2-39
ObjectInitialized.	2-39
ObjectName	2-39

ObjectPage	2-40
ObjectParent	2-40
ObjectType	2-40
ObjectVersion	2-40
ParentDataKey	2-40
PassThroughLinks	2-41
PhysicalObjectName	2-41
PhysicalVersion	2-41
PropertyDialog	2-42
QueryObject	2-42
RunAttribute	2-42
ServerFileName	2-43
ServerOperatingMode	2-43
SupportedLinks	2-43
TranslatableProperties	2-43
UIBMode	2-43
UseRepository	2-44
UserProperty	2-44
 3. Visual Objects and Their Methods and Properties	 3-1
Base methods for visual objects	3-2
applyLayout	3-2
assignFocusedWidget	3-2
assignWidgetValue	3-3
assignWidgetValueList	3-4
blankWidget	3-5
disableObject	3-6
disableRadioButton	3-6
disableWidget	3-7
enableObject	3-8
enableRadioButton	3-8
enableWidget	3-9
formattedWidgetValue	3-10
formattedWidgetValueList	3-10
hideWidget	3-12
highlightWidget	3-12
initializeObject	3-14
resetWidgetValue	3-14
toggleWidget	3-15
viewWidget	3-16
widgetHandle	3-17
widgetIsBlank	3-17
widgetIsFocused	3-18
widgetIsModified	3-18

widgetIsTrue	3-19
widgetValue.	3-20
widgetValueList	3-21
Methods for data visualization objects	3-22
addRecord.	3-22
cancelObject	3-22
cancelRecord	3-23
canNavigate	3-24
collectChanges	3-24
confirmCancel	3-25
confirmCommit	3-25
confirmContinue	3-26
confirmDelete	3-26
confirmExit	3-26
confirmOk	3-27
confirmUndo	3-27
copyRecord	3-27
dataAvailable.	3-28
deleteRecord	3-28
disableFields	3-29
displayObjects	3-29
displayRecord	3-30
enableFields	3-30
initializeObject	3-30
IsUpdateActive	3-31
linkStateHandler	3-31
okObject	3-31
okToContinueProcedure	3-32
queryPosition	3-32
resetRecord	3-33
showDataMessages	3-33
showDataMessagesProcedure	3-34
updateMode	3-34
updateRecord	3-35
updateState	3-35
validateFields	3-35
Filtering methods for visual objects	3-36
applyFilter	3-36
blankField	3-37
blankFields	3-37
blankFillIn	3-38
createField	3-38
createLabel	3-39
createOperator	3-40
dataAvailable.	3-40

dataValue	3-41
deleteObjects	3-41
disableFields	3-41
enableFields	3-41
fieldLookup	3-42
fieldProperty	3-42
initializeObject	3-42
insertFieldProperty	3-43
removeSpace	3-43
resetFields	3-44
showDataMessages	3-44
unBlankFillin	3-44
unBlankLogical	3-45
Browser methods for visual objects	3-45
addRecord	3-45
applyCellEntry	3-46
applyEntry	3-46
assignMaxGuess	3-47
calcWidth	3-47
cancelNew	3-47
cancelRecord	3-48
colValues	3-48
copyRecord	3-49
dataAvailable	3-49
defaultAction	3-50
deleteComplete	3-50
deleteRecord	3-50
destroyObject	3-51
disableFields	3-51
displayFields	3-52
enableFields	3-52
fetchDataSet	3-53
filterActive	3-53
hasActiveAudit	3-53
hasActiveComments	3-54
initializeObject	3-54
launchFolderWindow	3-55
offEnd	3-55
offHome	3-55
onEnd	3-55
onHome	3-56
refreshBrowse	3-56
resetRecord	3-56
resizeBrowse	3-57
resizeObject	3-57

rowChanged	3-58
rowDisplay	3-58
rowVisible	3-58
searchTrigger	3-59
startSearch	3-59
stripCalcs	3-59
toolbar	3-60
updateRecord	3-60
updateState	3-61
updateTitle	3-61
ValueChanged	3-61
viewObject	3-62
Methods for SmartDataViewers	3-62
addRecord	3-62
cancelRecord	3-63
copyRecord	3-64
disableFields	3-65
displayFields	3-66
enableFields	3-68
fieldModified	3-68
initializeObject	3-69
linkState	3-70
linkStateHandler	3-70
toolbar	3-70
updateState	3-71
valueChanged	3-71
viewRecord	3-71
Methods for TreeView objects	3-72
addNode	3-72
deleteNode	3-73
deleteTree	3-74
disableObject	3-74
emptyTree	3-74
enableObject	3-74
isNodeExpanded	3-75
loadImage	3-75
populateTree	3-76
repositionObject	3-77
resizeObject	3-77
selectFirstNode	3-78
selectNode	3-78
showTVError	3-78
updateTree	3-79
Visual object properties	3-79
AllFieldHandles	3-80

AllFieldNames	3-80
ApplyActionOnExit	3-80
ApplyExitOnAction	3-81
AutoSort	3-81
BrowseHandle	3-81
CalcWidth	3-81
ClientRect	3-81
Col	3-81
color3DFace	3-82
color3DHighLight	3-82
color3DShadow	3-82
CreateHandles	3-82
CtrlFrameHandle	3-82
DataModified	3-83
DataObject	3-83
DataSignature	3-83
DefaultCharWidth	3-83
DefaultEditorLines	3-84
DefaultLayout	3-84
DefaultWidth	3-84
DisableOnInit	3-84
DisplayedField	3-84
DisplayFieldsSecurity	3-84
DisplayedTables	3-85
Down	3-85
Editable	3-85
EnabledObjFlds	3-85
EnabledObjFldsToDisable	3-85
EnabledObjHdls	3-86
EnabledFields	3-86
EnabledHandles	3-86
ExpandOnAdd	3-86
FieldColumn	3-87
FieldFormats	3-87
FieldHandles	3-87
FieldHelpIds	3-87
FieldsEnabled	3-87
FieldLabels	3-88
FieldOperatorStyles	3-88
FieldToolTips	3-88
FieldWidths	3-88
FilterActive	3-88
FilterTarget	3-89
FilterTargetEvents	3-89
FolderWindowToLaunch	3-89

FrameMinHeightChars	3-89
FrameMinWidthChars	3-90
FullRowSelect	3-90
GroupAssignHidden	3-90
GroupAssignSource	3-90
GroupAssignSourceEvents	3-91
GroupAssignTarget	3-91
GroupAssignTargetEvents	3-91
Height	3-91
HideOnInit	3-91
HideSelection	3-91
ImageHeight	3-92
ImageWidth	3-92
ILComHandle	3-92
Indentation	3-92
KeepChildPositions	3-92
LabelEdit	3-93
LayoutOptions	3-93
LayoutVariable	3-93
LineStyle	3-93
MaxWidth	3-93
MinHeight	3-94
MinWidth	3-94
ModifyFields	3-94
NextNodeKey	3-95
NodeExpanded	3-95
NumDown	3-95
ObjectEnabled	3-95
ObjectLayout	3-96
OLEDrag	3-96
OLEDrop	3-96
Operator	3-96
OperatorLongValues	3-96
OperatorStyle	3-96
OperatorViewAs	3-97
Property	3-97
QueryRowObject	3-98
RecordState	3-98
Refresh	3-99
ResizeHorizontal	3-99
ResizeVertical	3-99
RootNodeParentKey	3-99
Row	3-99
RowIdent	3-100
RowObject	3-100

SaveSource	3-100
Scroll	3-100
ScrollRemote	3-101
SearchField	3-101
SelectedNode.	3-101
ShowCheckBoxes	3-101
ShowRootLines	3-101
SingleSel	3-102
Sort	3-102
TableIOSource	3-102
TableIOSourceEvents	3-102
ToggleDataTargets.	3-102
ToolbarSource	3-103
ToolbarSourceEvents.	3-103
ToolTip	3-103
TreeDataTable	3-103
TreeStyle	3-104
TVControllerSource	3-104
TVControllerTarget.	3-104
TVControllerTargetEvents	3-104
UpdateTarget	3-105
UpdateTargetNames	3-105
UseBegins	3-105
UseContains.	3-105
ValidKey	3-105
VisualBlank	3-106
Width	3-106
WindowTitleField	3-106
Column properties for visual objects	3-106
 4. Container Objects and Their Methods and Properties	 4-1
Base methods for container objects	4-2
addDataObject	4-2
applyContextFromServer	4-2
assignContainedProperties	4-3
assignPageProperty	4-4
cancelObject	4-5
changePage	4-5
clearCache	4-6
confirmExit	4-7
confirmOk	4-7
constructObject	4-8
containedProperties	4-8
ContextandDestroy.	4-9

createObjects	4-9
confirmCancel	4-10
deletePage	4-11
destroyObject	4-11
disablePagesInFolder	4-12
enablePagesInFolder	4-12
fetchContainedData	4-12
findCacheItem	4-13
hidePage	4-13
initializeObject	4-14
initializeVisualContainer	4-14
initPages	4-14
isUpdateActive	4-15
notifyPage	4-16
obtainContextForServer	4-17
okObject	4-18
pageNTargets	4-18
passThrough	4-19
registerCacheItem	4-20
removePageNTarget	4-20
resizeWindow	4-21
selectPage	4-21
targetPage	4-22
toolbar	4-22
updateActive	4-23
viewObject	4-24
viewPage	4-24
Methods for SmartBusiness container objects	4-25
addDataTarget	4-25
addNavigationSource	4-26
addQueryWhere	4-26
addRow	4-27
appendContainedObjects	4-27
assignCurrentMappedObject	4-28
assignMaxGuess	4-28
assignQuerySelection	4-28
cancelNew	4-29
cancelRow	4-29
canNavigate	4-30
colValues	4-30
commitData	4-31
commitTransaction	4-31
confirmContinue	4-31
copyRow	4-32
currentMappedObject	4-32

dataAvailable	4-32
dataObjectHandle.	4-33
deleteComplete	4-33
deleteRow	4-33
destroyServerObject.	4-33
endClientDataRequest	4-34
fetchBatch	4-34
fetchContainedData	4-34
fetchContainedRows	4-35
fetchDOProperties	4-35
fetchFirst	4-35
fetchLast	4-36
fetchNext	4-36
initDataObjectOrdering.	4-36
fetchPrev	4-36
findRowWhere	4-37
initializeObject	4-38
initializeServerObject	4-38
isUpdatePending	4-38
newDataObjectRow	4-39
openQuery	4-39
postCreateObjects	4-39
prepareErrorsForReturn	4-40
prepareQueriesForFetch	4-40
queryPosition	4-41
refreshBrowse	4-41
registerLinkedObjects.	4-41
registerObject	4-42
remoteCommitTransaction	4-42
remoteFetchContainedData	4-42
remoteSendRows.	4-43
removeQuerySelection	4-45
repositionRowObject	4-45
resetQuery	4-46
restartServerObject	4-46
serverCommitTransaction	4-47
serverContainedSendRows	4-48
serverFetchContainedData	4-49
serverFetchContainedRows	4-49
serverFetchDOProperties.	4-50
setPropertyList	4-50
startServerObject	4-50
submitRow	4-51
undoTransaction.	4-51
updateState	4-51

Methods for TreeView controller container objects	4-52
initializeObject	4-52
postCreateObjects	4-52
showTVCErrors	4-52
updateState	4-53
Container object properties	4-53
BlockDataAvailable	4-53
CallerObject	4-54
CallerProcedure	4-54
CallerWindow	4-54
CascadeOnBrowse	4-54
CommitSource	4-54
CommitSourceEvents	4-55
CommitTarget	4-55
CommitTargetEvents	4-55
ContainedDataColumns	4-55
ContainedDataObjects	4-56
ContainerTarget	4-56
ContainerTargetEvents	4-56
ContextAndInitialize	4-56
CurrentPage	4-56
DataHandle	4-57
DataObjectNames	4-57
DataObjectOrdering	4-57
DataQueryBrowsed	4-57
DynamicSDOProcedure	4-58
FetchOnOpen	4-58
FilterSource	4-58
InitialPageList	4-58
InMessageTarget	4-58
InstanceNames	4-58
LastCommitErrorKeys	4-59
LastCommitErrorType	4-59
MasterDataObject	4-60
MultiInstanceActivated	4-60
MultiInstanceSupported	4-60
NavigationSource	4-60
NavigationSourceEvents	4-60
NavigationTarget	4-60
ObjectMapping	4-61
OutMessageTarget	4-61
PageNTarget	4-61
PageSource	4-61
PrimarySdoTarget	4-61
QueryPosition	4-62

RouterTarget	4-62
RowObjectState	4-62
RunDataLogicProxy	4-62
RunDOOptions	4-64
SdoForeignFields	4-65
TopOnly	4-65
UpdateActive	4-65
UpdateSource	4-65
WaitForObject	4-65
WindowFrameHandle	4-66
Column properties for container objects	4-66
5. Query Objects and Their Methods and Properties	5-1
Base methods for query objects	5-2
addNotFoundMessage	5-2
addQueryWhere	5-2
assignDBRow	5-3
assignQuerySelection	5-4
batchServices	5-5
bufferCopyDBToRO	5-6
ColumnPhysicalColumn	5-6
ColumnPhysicalTable	5-7
closeQuery	5-7
colValues	5-8
confirmCommit	5-8
confirmContinue	5-8
confirmUndo	5-9
dataAvailable	5-10
dbColumnName	5-11
dbColumnHandle	5-11
defineDataObject	5-12
deleteRecordStatic	5-13
fetchFirst	5-13
fetchLast	5-13
fetchNext	5-14
fetchPrev	5-14
fetchFirstBatch	5-14
fetchNextBatch	5-14
fetchPrevBatch	5-14
fetchLastBatch	5-15
fetchCurrentBatch	5-15
filterContainerHandler	5-15
firstBufferName	5-16
firstRowIds	5-16

indexInformation	5-17
initProps	5-17
initializeObject	5-17
insertExpression	5-18
newQueryString	5-19
newQueryValidate	5-20
newQueryWhere	5-21
newWhereClause	5-21
openQuery	5-22
prepareQuery	5-22
removeForeignKey	5-23
removeQuerySelection	5-23
resolveBuffer	5-24
rowidWhere	5-24
rowidWhereCols	5-25
startFilter	5-25
transferDBRow	5-26
whereClauseBuffer	5-26
Methods for data query objects	5-27
addRow	5-27
askQuestion	5-27
batchRowAvailable	5-28
beginTransactionValidate	5-28
cancelRow	5-29
canNavigate	5-29
clientSendRows	5-30
closeQuery	5-31
colStringValue	5-31
colValues	5-32
Commit	5-32
commitData	5-33
commitTransaction	5-33
copyRow	5-33
createData	5-34
createObjects	5-34
createRow	5-34
dataAvailable	5-35
dataContainerHandle	5-36
deleteData	5-36
deleteRow	5-37
describeSchema	5-37
destroyObject	5-37
destroyServerObject	5-38
doBuildUpd	5-38
doCreateUpdate	5-39

doEmptyTempTable	5-39
doReturnUpd	5-40
doUndoDelete	5-40
doUndoRow	5-41
doUndoTrans	5-41
doUndoUpdate	5-41
endClientDataRequest	5-41
endTransactionValidate	5-42
exportData	5-42
fetchBatch	5-43
fetchFirst	5-43
fetchLast	5-44
fetchNext	5-44
fetchPrev	5-44
fetchRow	5-45
fetchRowIdent	5-45
findRow	5-46
findRowWhere	5-47
firstRowIds	5-48
hasActiveAudit	5-48
hasActiveComments	5-48
hasForeignKeyChanged	5-48
hasOneToOneTarget	5-49
initializeObject	5-49
initializeLogicObject	5-49
initializeServerObject	5-50
isUpdateActive	5-50
isUpdatePending	5-50
linkState	5-51
LinkStateHandler	5-51
newRowObject	5-52
obtainContextForServer	5-52
openDataQuery	5-52
openQuery	5-53
postTransactionValidate	5-53
prepareErrorsForReturn	5-53
preTransactionValidate	5-54
printToCrystal	5-54
pushTableAndValidate	5-54
refreshRow	5-55
remoteCommit	5-55
remoteSendRows	5-56
repositionRowObject	5-57
resortQuery	5-58
rowAvailable	5-58

rowBatchAvailable	5-59
rowObjectValidate	5-59
rowValues	5-60
saveContextAndDestroy	5-61
sendRows	5-62
serverCommit	5-63
serverSendRows	5-64
startServerObject	5-65
submitCommit	5-65
submitForeignKey	5-66
submitRow	5-67
synchronizeProperties	5-68
tableOut	5-68
transferToExcel	5-69
undoClientUpdate	5-70
undoTransaction	5-70
updateAddQueryWhere	5-70
updateData	5-71
updateManualForeignFields	5-71
updateQueryPosition	5-72
updateRow	5-72
updateState	5-73
Query object properties	5-73
AssignList	5-74
AutoCommit	5-74
CacheDuration	5-74
CalculatedColumns	5-74
CheckCurrentChanged	5-74
CheckLastOnOpen	5-75
ClientProxyHandle	5-75
CurrentRowModified	5-75
DataColumns	5-76
DataColumnsByTables	5-76
DataDelimiter	5-76
DataFieldDefs	5-76
DataHandle	5-77
DataLogicObject	5-77
DataModified	5-77
DatalsFetched	5-77
DataLogicProcedure	5-78
DataQueryBrowsed	5-78
DataQueryString	5-78
DataReadBuffer	5-78
DataReadColumns	5-78
DataReadHandler	5-79

DataSignature	5-79
DbNames	5-79
DestroyStateless	5-80
DisconnectAppServer.	5-80
EnabledObjFldsToDisable	5-80
EnabledTables	5-81
FillBatchOnRepos	5-81
FilterWindow.	5-81
FirstResultRow	5-81
FirstRowNum	5-81
FilterActive	5-82
FilterAvailable.	5-82
ForeignFields	5-82
ForeignValues	5-82
GroupAssignSource	5-82
GroupAssignSourceEvents	5-83
GroupAssignTarget	5-83
GroupAssignTargetEvents	5-83
IndexInformation.	5-83
InternalEntries	5-83
KeyFields	5-84
LastCommitErrorKeys	5-84
LastCommitErrorType	5-84
LastDbRowIdent.	5-85
LastCommitErrorType	5-85
LastResultRow	5-86
LastRowNum	5-86
LogicBuffer	5-86
ManualAddQueryWhere.	5-86
ManualAssignQuerySelection	5-87
ManualSetQuerySort	5-87
NavigationSource	5-87
NavigationSourceEvents	5-87
NewRow	5-88
OpenOnInit.	5-88
OpenQuery.	5-88
PhysicalTable.	5-89
PropertyList	5-89
QueryContainer	5-89
QueryHandle	5-89
QueryOpen	5-90
QueryPosition.	5-90
QueryRowIden	5-90
QuerySort.	5-90
QueryString	5-91

QueryWhere	5-91
RebuildOnRepos.	5-92
RowIdent.	5-92
RowObjectState	5-92
RowObjectTable	5-92
RowObjUpd.	5-92
RowObjUpdTable	5-93
RowsToBatch	5-93
SchemaLocation	5-93
ServerSubmitValidation.	5-93
ShareData	5-94
Tables	5-94
ToggleDataTargets	5-94
UseDBQualifier	5-95
UpdateFromSource.	5-95
UpdatableColumns	5-95
UpdatableColumnsByTable	5-96
WordIndexedFields	5-96
Column properties for query objects	5-96
6. Toolbar Objects and Their Methods and Properties.	6-1
Methods for toolbar object actions	6-2
canFindAction	6-2
canFindCategory.	6-2
categoryLink	6-2
checkRule	6-3
defineAction	6-3
displayActions.	6-4
initAction	6-4
initializeObject.	6-4
Panel methods for toolbar objects	6-4
activeTarget	6-4
countButtons	6-5
disableActions.	6-5
enableActions	6-6
enableObject.	6-6
hasActiveGATarget.	6-6
initializeObject.	6-7
linkState	6-8
loadPanel	6-8
onChoose	6-8
queryPosition	6-9
resetCommit	6-9
resetNavigation	6-9

resetTableio	6-9
resetTargetActions	6-10
resizeObject	6-10
sensitizeActions	6-11
targetActions	6-11
updateState	6-11
viewHideActions	6-12
Toolbar methods for toolbar objects	6-12
actionCanRun	6-12
actionCaption	6-13
actionCategoryIsHidden	6-13
actionChecked	6-14
actionLabel	6-14
actionPublishCreate	6-14
actionTarget	6-15
actionTooltip	6-15
buildAllMenus	6-16
categoryActions	6-16
constructMenu	6-16
constructToolbar	6-16
createMenuTable	6-17
createMenuTable2	6-18
createToolbar	6-18
filterState	6-19
hideObject	6-19
imageName	6-19
initializeObject	6-20
insertMenu	6-20
linkRuleBuffer	6-21
linkState	6-21
modifyDisabledActions	6-22
onChoose	6-22
onMenuDrop	6-23
onValueChanged	6-23
queryPosition	6-23
repositionObject	6-24
resetTargetActions	6-24
resetToolbar	6-24
resizeObject	6-25
retrieveBandsAndActions	6-25
rowObjectState	6-26
runInfo	6-26
storePendingSensitivity	6-27
targetActions	6-27
updateActive	6-27

updateCategoryLists	6-28
updateState	6-28
updateStates	6-29
viewHideActions	6-29
viewObject	6-29
Toolbar object properties	6-30
AvailMenuActions	6-30
AvailToolBarActions	6-30
AvailToolBarBands	6-30
BoxRectangle	6-31
BoxRectangle2	6-31
ButtonCount	6-31
DeactivateTargetOnHide	6-31
DisabledActions	6-32
EdgePixels	6-32
HiddenActions	6-32
HiddenMenuBands	6-32
HiddenToolBarBands	6-32
Image	6-33
ImagePath	6-33
MarginPixels	6-33
Menu	6-33
MenuMergeOrder	6-33
MinHeight	6-34
MinWidth	6-34
NavigationTargetEvents	6-34
NavigationTargetName	6-34
PanelFrame	6-34
PanelLabel	6-35
PanelState	6-35
PanelType	6-35
SecuredTokens	6-35
ShowBorder	6-35
StaticPrefix	6-35
TableIOTarget	6-36
TableIOTargetEvents	6-36
TableIOType	6-36
ToolBar	6-36
ToolBarAutoSize	6-36
ToolBarBands	6-37
ToolBarDrawDirection	6-37
ToolBarHeightPxl	6-37
ToolBarInitialState	6-37
ToolBarTargetEvents	6-37
ToolBarWidthPxl	6-38

ToolMarginPxI	6–38
Action properties for toolbar objects	6–38
7. Field Objects and Their Methods and Properties	7–1
Base methods for field objects	7–2
initializeObject	7–2
resizeObject	7–2
Methods for select field objects	7–3
anyKey	7–3
browseHandler	7–3
buildList	7–3
createLabel.	7–4
dataAvailable	7–4
destroyObject	7–5
destroySelection.	7–5
disableButton	7–5
disableField	7–5
disable_UI	7–6
enableButton	7–6
enableField.	7–6
endMove	7–6
enterSelect	7–7
formattedValue.	7–7
hideObject	7–7
initializeBrowse	7–8
initializeObject	7–8
initializeSelection	7–8
leaveSelect.	7–8
queryOpened	7–9
refreshObject	7–9
repositionDataSource.	7–9
resizeObject	7–10
rowSelected	7–10
valueChanged	7–10
viewObject	7–11
Base methods for combo and lookup objects	7–11
notifyChildFields.	7–11
retrieveData	7–12
returnParentFieldValues.	7–12
Methods for combo field objects	7–13
anyKey	7–13
createDataSource	7–13
createLabel.	7–13
destroyCombo	7–14

destroyObject	7-14
disableField	7-14
disable_UI	7-14
displayField	7-15
displayCombo	7-15
enableField	7-15
enterCombo	7-16
endMove	7-16
hideObject	7-16
initializeCombo	7-16
insertExpression	7-17
leaveCombo	7-17
newQueryString	7-18
newWhereClause	7-19
prepareField	7-19
refreshChildDependancies	7-20
refreshCombo	7-20
resizeObject	7-20
returnParentFieldValues	7-21
showQuery	7-21
valueChanged	7-21
whereClauseBuffer	7-22
viewObject	7-22
Methods for lookup field objects	7-22
anyKey	7-22
createLabel	7-23
destroyLookup	7-23
destroyObject	7-23
disableButton	7-23
disableField	7-24
disable_UI	7-24
displayField	7-24
displayLookup	7-24
enableButton	7-25
enableField	7-25
endMove	7-25
enterLookup	7-25
hideObject	7-26
initializeBrowse	7-26
initializeLookup	7-26
insertExpression	7-27
leaveLookup	7-27
newQueryString	7-28
newWhereClause	7-29
prepareField	7-29

resizeObject	7-30
returnParentFieldValues.	7-30
rowSelected	7-30
translateBrowseColumns	7-31
valueChanged	7-31
viewObjec.	7-31
whereClauseBuffer.	7-32
Field object properties	7-32
BaseQueryString	7-33
BrowseFields	7-33
BrowseFieldFormats	7-33
BrowseFields	7-33
BrowseTitle.	7-33
BuildSequence	7-34
CancelBrowseOnExit	7-34
ChangedEvent	7-34
ComboBuffer	7-34
ComboDelimiter	7-34
ComboFlag	7-35
ComboFlagValue	7-35
ComboHandle	7-35
ComboQuery	7-36
ComboSort	7-36
CurrentDescValue	7-36
CurrentKeyValue	7-36
DataModified	7-37
DataSourceFilter	7-37
DataSourceName.	7-37
DataValue.	7-37
DefineAnyKeyTrigger	7-37
DescSubstitute	7-38
DisplayDataType	7-38
DisplayField	7-38
DisplayFormat	7-38
DisplayValue	7-38
EdgePixels	7-39
EnableField	7-39
EnableOnAdd.	7-39
ExitBrowseOnAction.	7-39
FieldEnabled	7-39
FieldHidden	7-40
FieldLabel.	7-40
FieldName	7-40
FieldToolTip	7-40
FlagValue	7-40

Format	7-40
HelpId	7-41
InnerLines	7-41
KeyDataType	7-41
KeyField	7-41
KeyFormat.	7-41
Label	7-41
LabelHandle	7-42
LinkedFieldDataTypes	7-42
LinkedFieldFormats.	7-42
ListItemPairs	7-42
LookupBuffer.	7-42
LookupFilterValue	7-43
LookupHandle.	7-43
LookupImage	7-43
LookupQuery.	7-43
MaintenanceObject	7-44
MaintenanceSDO	7-44
NumRows	7-44
Optional	7-44
OptionalBlank	7-44
OptionalString	7-45
ParentField	7-45
ParentFilterQuery	7-45
PopupOnAmbiguous	7-45
PopupOnUniqueAmbiguous	7-46
PopupOnNotAvail	7-46
QueryTables	7-46
RefreshList	7-46
RepositionDataSource	7-47
RowsToBatch	7-47
SavedScreenValue	7-47
SDFFilename	7-47
SDFTemplate	7-48
Secured	7-48
StartBrowseKeys.	7-48
ToolTip	7-48
UsePairedList	7-48
ViewAs	7-49
ViewerLinkedFields	7-49
ViewerLinkedWidgets	7-49
Field object column properties	7-50

8. Messaging Objects and Their Methods and Properties	8-1
Base methods for messaging objects	8-2
destroyObject	8-2
errorHandler	8-2
initializeObject	8-2
Methods for consumer messaging objects	8-3
assignUnsubscribe	8-3
createConsumers	8-3
defineDestination	8-4
destroyObject	8-4
errorHandler	8-4
initializeObject	8-5
messageHandler	8-5
processDestinations	8-5
startWaitFor	8-6
stopHandler	8-6
Methods for producer messaging objects	8-6
destroyObject	8-6
initializeObject	8-7
replyHandler	8-7
sendMessage	8-7
Method for message handler objects	8-8
sendMessage	8-8
Methods for XML messaging objects	8-8
assignAttribute	8-8
assignNodeValue	8-9
createAttribute	8-9
createDocument	8-9
createElement	8-10
createNode	8-10
createText	8-11
deleteDocument	8-11
destroyObject	8-11
nodeHandle	8-12
nodeType	8-12
ownerElement	8-12
parentNode	8-12
processCDATASection	8-13
processComment	8-13
processDocument	8-13
processElement	8-14
processRoot	8-14
processText	8-14
receiveHandler	8-15
receiveReplyHandler	8-15

sendHandler	8–15
sendReplyHandler	8–16
Methods for busines to business messaging Objects	8–16
callOutParams	8–16
characterValue	8–17
createSchemaAttributes	8–17
createSchemaChildren	8–17
createSchemaPath	8–18
dataSource	8–18
defineMapping	8–18
destroyObject	8–19
endDocument	8–19
endElement	8–19
findDataRow	8–20
initializeObject	8–20
loadProducerSchema	8–20
loadSchema	8–21
mapNode	8–21
NotFoundError	8–22
numParameters	8–22
processMappings	8–22
processMessages	8–23
produceAttributes	8–23
produceChildren	8–24
produceDocument	8–24
receiveHandler	8–24
rowNotFoundError	8–25
schemaField	8–25
sendHandler	8–26
sendMessage	8–26
startDataRow	8–26
startElement	8–27
storeNodeValue	8–27
storeParameterNode	8–28
storeParameterValue	8–28
targetProcedure	8–29
Methods for router messaging objects	8–29
createDocument	8–29
initializeObject	8–30
internalSchemaFile	8–30
obtainInMsgTarget	8–30
processFileRefs	8–31
routeBytesMessage	8–31
routeDocument	8–31
routeMessage	8–32

startB2BObject	8-32
Message object properties	8-33
ClientID	8-33
ConsumerSchema	8-33
ContextForServer	8-33
CurrentMessage	8-34
CurrentMessageId	8-34
Destination	8-34
DestinationList	8-34
Destinations	8-34
DirectionList	8-35
DocumentElement	8-35
DocumentHandle	8-35
DocumentInitialized	8-35
Domain	8-35
DTDPublicId	8-36
DTDPublicIdList	8-36
DTDSysId	8-36
DTDSysIdList	8-36
ExternalRefList	8-36
InMessageSource	8-36
InternalRefList	8-37
JMSpartition	8-37
JMSpassword	8-37
JMSsession	8-37
JMSuser	8-38
LoadedByRouter	8-38
LogFile	8-38
MapNameProducer	8-38
MapObjectProducer	8-38
MapTypeProducer	8-39
MessageType	8-39
NameList	8-39
NamespaceHandle	8-39
NewNode	8-39
OutMessageSource	8-40
Persistency	8-40
PingInterval	8-40
Priority	8-40
PromptLogin	8-40
ReplyReqList	8-41
ReplyRequired	8-41
ReplySelector	8-41
ReplySelectorList	8-41
RouterSource	8-41

SchemaHandle	8-42
SchemaList	8-42
SchemaManager	8-42
Selectors	8-42
ShutDownDest	8-42
Subscriptions	8-42
SupportedMessageTypes	8-43
TargetNameSpace	8-43
TimeToLive	8-43
TypeName	8-43
UseDTD	8-43
ValidateOnLoad	8-44
Waiting	8-44
 9. Alphabetical Listing Of WebSpeed-specific API Routines	 9-1
addColumnLink	9-2
addContextFields	9-5
addSearchCriteria	9-5
addTDModifier	9-6
anyMessage	9-6
AddMode	9-7
assignColumnFormat	9-7
assignColumnHelp	9-8
assignColumnLabel	9-8
assignColumnWidth	9-9
assignExtentAttribute	9-9
assignFields	9-10
assignFields	9-10
assignTDModifier	9-11
Attribute	9-12
bufferHandle	9-12
bufferHandle	9-13
columnDataType	9-13
columnFormat	9-13
columnHandle	9-14
columnHelp	9-14
columnHTMLName	9-14
columnHTMLName	9-15
columnLabel	9-16
columnReadOnly	9-16
columnStringValue	9-16
columnTable	9-17
columnTDModifier	9-17
columnValMsg	9-18

constructObject	9-18
dataAvailable	9-19
deleteBuffer	9-19
deleteOffsets	9-20
deleteRow	9-20
destroy	9-20
destroyDataObject	9-21
destroyObject	9-21
destroyObject	9-21
disconnectObject	9-21
dispatchUtilityProc	9-22
displayFields.	9-22
enableFields.	9-23
exclusiveLockBuffer	9-24
extentAttribute	9-24
extentValue	9-25
fetchCurrent	9-25
fetchFirst	9-25
fetchLast	9-26
fetchLast	9-26
fetchNext	9-26
fetchNext	9-27
fetchPrev	9-27
fetchPrev	9-27
fieldExpression	9-28
findRecords	9-28
getAttribute	9-29
getContextFields	9-29
getCurrentPage	9-30
getCurrentRowids.	9-30
getDeleteTables	9-30
getForeignFieldList	9-31
getFrameHandle.	9-31
getNavigationMode	9-31
getNextHtmlField	9-32
getQueryEmpty	9-33
getQueryWhere	9-33
getRowids.	9-33
getSearchColumns.	9-33
getServerConnection	9-34
getTableRowids	9-34
getTableRows	9-34
getTables	9-34
getTables	9-35
getUpdateMode	9-35

getWebState	9-35
getWebTimeout	9-36
getWebTimeRemaining.	9-36
getWebToHdlr	9-36
htmAssociate.	9-37
HTMLAlert	9-37
HTMLColumn	9-38
HTMLSetFocus	9-38
HTMLTable	9-39
initializeObject.	9-39
inputFields.	9-39
joinExternalTables.	9-40
joinForeignFields.	9-41
lockRow	9-41
openQuery	9-42
outputFields.	9-42
pageBackward	9-42
processWebRequest.	9-43
processWebRequest	9-44
processWebRequest.	9-44
readOffsets	9-45
removeEntry	9-45
reOpenQuery	9-46
rowidExpression	9-46
setAddMode	9-46
setAppService.	9-47
set-attribute-list	9-47
setBuffers	9-48
setColumns	9-48
setContextFields	9-48
setCurrentRowids	9-49
setDeleteTables	9-49
setExternalJoinList	9-50
setExternalTableList	9-50
setExternalTables	9-51
setExternalWhereList	9-52
setForeignFieldList	9-52
setFrameHandle	9-53
setLinkColumns.	9-53
setLinkColumns.	9-53
setQueryWhere	9-54
setSearchColumns	9-54
setServerConnection.	9-54
setTableModifier	9-55
setTableRows	9-55

setUpDateMode	9-56
setUseColumnLabels	9-56
setWebState	9-57
setWebToHdlr	9-57
showDataMessages	9-58
startDataObject	9-58
timingOut	9-58
urlJoinParams	9-59
urlLink	9-60
validateColumns	9-60
validateColumnValue	9-61
Progress Dynamics Call Wrapper	A-1
Invoking the dynamics call wrapper using dynlaunch.i	A-2
Invoking the dynamics call wrapper using a single-entry point	A-3
callstring.p	A-4
callstringtt.p	A-5
calltable.p	A-7
calltablett.p	A-10
Temp-table include files	A-11
calltables.i	A-11
callttparam.i	A-12
Temp-table types	A-14
Position native data type table	A-14
Position character table	A-16
Name native data type table	A-17
Name character table	A-19
Invoking the dynamics call wrapper at the API level	A-20
API Reference	A-26
callstring.p Procedure	A-26
callstringtt.p procedure	A-28
calltable.p procedure	A-31
calltablett.p procedure	A-33
cleanupCall procedure	A-36
determineTableType function	A-36
InvokeCall procedure	A-37
obtainCallInfo function	A-39
obtainParamPropValue function	A-39
obtainProcHandle function	A-40
setupTTFFromSig function	A-40
setupTTFFromString function	A-41
setupTTFFromTable function	A-43
Index	Index-1

Figures

Figure 1–1:	ADM2 class tree diagram showing inheritance relationships	1–4
Figure 1–2:	Commented source code of GET pseudo-function	1–29
Figure A–1:	Position native data type table definition	A–14
Figure A–2:	Position character table type definition	A–16
Figure A–3:	Name native data type table definition	A–17
Figure A–4:	API calls required to invoke a dynamics call	A–21

Tables

Table 1–1:	SmartObjects and their files	1–5
Table 1–2:	Class and custom files for SmartObjects	1–8
Table 1–3:	Class and custom files for SmartContainers	1–8
Table 1–4:	Class and custom files for AppServer	1–9
Table 1–5:	Class and custom files for SmartBusinessObjects	1–10
Table 1–6:	Class and custom files for SmartB2BObjects	1–11
Table 1–7:	Class and custom files for MsgHandler	1–11
Table 1–8:	Class and custom files for XML	1–12
Table 1–9:	Class and custom files for SmartCombo	1–12
Table 1–10:	Class and custom files for SmartDataBrowser	1–13
Table 1–11:	Class and custom files for SmartDataField	1–14
Table 1–12:	Custom and class files for SmartDataObjects	1–15
Table 1–13:	Custom and class files for queries	1–15
Table 1–14:	Class and custom files for SmartDataViewer	1–16
Table 1–15:	Class and custom files for SmartDialog	1–17
Table 1–16:	Class and custom files for SmartFilter	1–18
Table 1–17:	Class and custom files for SmartFolder	1–18
Table 1–18:	Class and custom files for SmartFrame	1–19
Table 1–19:	Class and custom files for SmartLookup	1–20
Table 1–20:	Class and custom files for SmartPanel	1–20
Table 1–21:	Class and custom files for SmartProducer	1–22
Table 1–22:	Class and custom files for Consumer	1–22
Table 1–23:	Class and custom files for messaging	1–23
Table 1–24:	Class and custom files for SmartRouter	1–23
Table 1–25:	Class and custom files for SmartSelect	1–24
Table 1–26:	Class and custom files for SmartSender and SmartReceiver	1–24
Table 1–27:	Class and custom files for SmartToolbar	1–25
Table 1–28:	Class and custom files for SmartWindow	1–26
Table 3–1:	Temp-table includes for fields defined in TVController	3–72
Table 3–2:	Column properties for visual objects	3–107
Table 4–1:	Column properties for container objects	4–67
Table 5–1:	Column properties for query objects	5–97
Table 6–1:	Action properties for toolbar objects	6–38
Table 7–1:	Column properties for field objects	7–50

Examples

Example A-1: Example server-side procedure	A-5
Example A-2: Making a call to the obtainCustomerData procedure	A-6
Example A-3: Using calltable.p to invoke a call	A-8
Example A-4: Using calltable.p to invoke a call and to instantiate calls.p	A-9
Example A-5: Using callstringtt.p to make a call	A-10
Example A-6: Using the callttparam.i include file	A-13
Example A-7: Parameters stored in a position native data type table	A-15
Example A-8: Parameters specified in iParamNo	A-15
Example A-9: Parameters stored in a position character table type	A-16
Example A-10: Name native data type table	A-18
Example A-11: Name character table definition	A-19
Example A-12: Steps for making dynamics calls	A-22
Example A-13: Creating the parameter holder temp-table	A-23
Example A-14: Deriving parameters from the signature	A-25
Example A-15: Deriving parameters from a string to create a dynamics temp-table . .	A-26

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 for Progress ADM2 application developers. Developers who use the ADM2 need a strong understanding of the underlying Progress 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

[SmartObjects and Their Methods and Properties](#)

Lists and describes the methods (internal procedures and functions) and properties used for the base ADM2 SmartObjects.

[Visual Objects and Their Methods and Properties](#)

Lists and describes the methods (internal procedures and functions) and properties used for visualization SmartObjects.

[Container Objects and Their Methods and Properties](#)

Lists and describes the methods (internal procedures and functions) and properties used for container SmartObjects.

[Query Objects and Their Methods and Properties](#)

Lists and describes the methods (internal procedures and functions) and properties used for query SmartObjects.

[Toolbar Objects and Their Methods and Properties](#)

Lists and describes the methods (internal procedures and functions) and properties used for toolbar SmartObjects.

[Field Objects and Their Methods and Properties](#)

Lists and describes the methods (internal procedures and functions) and properties used for field SmartObjects.

[Messaging Objects and Their Methods and Properties](#)

Lists and describes the methods (internal procedures and functions) and properties used for messaging SmartObjects.

[Alphabetical Listing Of WebSpeed-specific API Routines](#)

Provides a description, including calling sequence, of each of the routines in the WebSpeed® super-procedure files.

[Progress Dynamics Call Wrapper](#)

This appendix provides information about the Progress Dynamic Call Wrapper. The Progress Dynamic Call Wrapper provides an efficient way to dynamically invoke code with parameter lists that are defined at run time.

How to use this manual

This manual is organized based on SmartObject type and each chapter lists and describes the methods and properties relevant to the specific SmartObjects type. Depending on the object type you want to create, refer to the chapter that covers that specific type of object.

For each chapter that describes properties that you can retrieve and set, the chapter provides a description of the **get** and **set** functions and identifies which properties can be read and which can be set.

For objects that use column or actions properties, these properties are listed in a table format and the chapter provides information about how to **assign** values for a property and identifies which properties can be read and which can be set.

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
 - Names of user interface elements
- *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 hyphen. 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 ... ] }
    }
```

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)
```

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**).

ADM2 SmartObject API Reference

The behavior of each type of Progress® SmartObject™ is defined by a set of methods (procedures and functions) based on one or more classes in 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 methods and properties 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 ADM2 Guide*. For information about creating applications using SmartObjects, see the *Progress AppBuilder Developer's Guide*. For information about SmartObjects in a Progress Dynamics® environment, see [Progress Dynamics Programming Handbook](#) and [Progress Dynamics Developer's Guide](#).

Paths to source files

The default paths to the files defining the SmartObjects are:

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

The default paths to the ADM2 files supporting WebSpeed are:

- **Class Source Path** — `src\web2`
- **Class Rcode Path** — `web2`
- **Class Template Path** — `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 accidentally 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*, *classnamecustom.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*, *classnamecustom.p*** — The primary and custom super-procedure files. These files contain the source code for most of the routines that define the class.

- ***classnameprop.i***, ***classnamepropcustom.i*** — The primary and custom property-definition files.
- ***classnameprto.i***, ***classnameprtocustom.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.
- ***classnameexclcustom.i*** — The (custom) exclusions file. This file defines EXCLUDE-*identifier* preprocessor variables that exclude routines from the super procedure.
- ***classnamedefscustom.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 ADM2 Guide*.

ADM class hierarchy

Figure 1–1 illustrates the hierarchy of relationships between the ADM classes used to create the objects. All Objects begin with the base Smart Object and from this object, you extend classes to create the other objects.

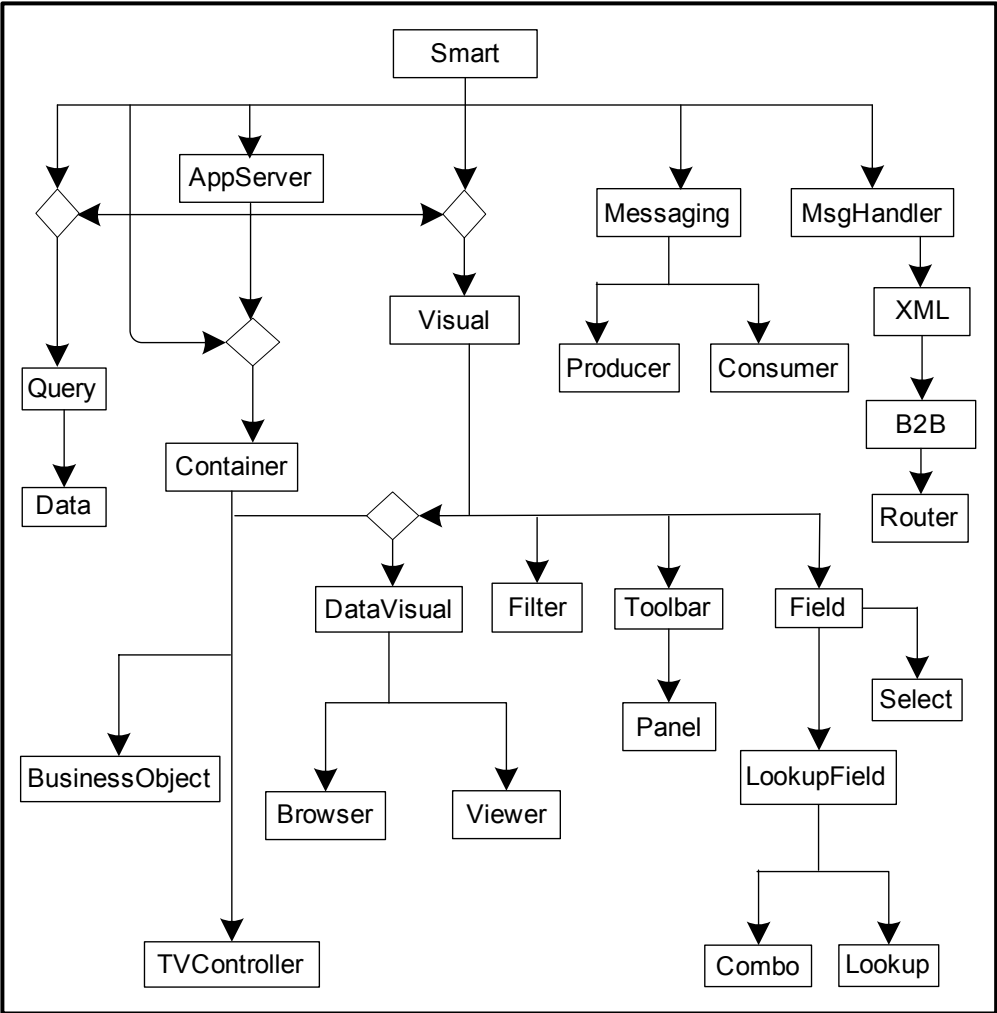


Figure 1–1: ADM2 class tree diagram showing inheritance relationships

SmartObjects, templates, and super-procedure hierarchies

[Table 1–1](#) summarizes 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 1–1: SmartObjects and their files

(1 of 3)

SmartObject	Super-procedure files	Template files
SmartObject	smart.p	smart.w
SmartContainer™	smart.p containr.p	cntnrtempl.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 lookupfield.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

Table 1–1: SmartObjects and their files*(2 of 3)*

SmartObject	Super-procedure files	Template files
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
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 lookupfield.p lookup.p	dynlookup.w

Table 1–1: SmartObjects and their files*(3 of 3)*

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

SmartObject

The SmartObject is the base object used for defining other objects. It has no special class identity of its own but is an expression of the class Smart.

The file `src/adm2/smart.p` is the super procedure supporting the base SmartObject. [Table 1–2](#) lists the class and custom files related to `smart.p`.

Table 1–2: Class and custom files for SmartObjects

Class files		Customer files	
Definition	<code>smart.cld</code>	Super	<code>smartcustom.p</code>
Method	<code>smart.i</code>	Method	<code>smartcustom.i</code>
Property	<code>smrtprop.i</code>	Property	<code>smrtpropcustom.i</code>
Prototype	<code>smrtprto.i</code>	Prototype	<code>smrtprtocustom.i</code>
Template	<code>smart.w</code>	Exclude	<code>smartexclcustom.i</code>
Other	<code>admmgs.i</code>	Instance	<code>smartdefscustom.i</code>

SmartContainer

The 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 but is an expression of the Container class and uses `containr.p` as its super-procedure file. [Table 1–3](#) lists the class and custom files related to `containr.p`.

Table 1–3: Class and custom files for SmartContainers (1 of 2)

Class files		Customer files	
Definition	<code>containr.cld</code>	Super	<code>containrcustom.p</code>
Method	<code>containr.i</code>	Method	<code>containrcustom.i</code>

Table 1–3: Class and custom files for SmartContainers

(2 of 2)

Class files		Customer files	
Property	cntnprop.i	Property	cntnpropcustom.i
Prototype	cntnprto.i	Prototype	cntnprtocustom.i
Template	cntnrsimpl.w	Exclude	containrexclcustom.i
–	–	Instance	containrdefscustom.i

Examples: AppServer class

If the macro {&APP-SERVER-VARS} is defined, the Container and Query classes 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. [Table 1–4](#) lists the class and custom files related to appserver.p.

Table 1–4: Class and custom files for AppServer

Class files		Customer 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 `src/adm2/sbo.p` is the super procedure file for the SBO class. [Table 1–5](#) lists the class and custom files related to `sbo.p`.

Table 1–5: Class and custom files for SmartBusinessObjects

Class files		Customer 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	updtbledefs.i updparam.i updtblecase.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 `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. [Table 1–6](#) lists the class and custom files related to `b2b.p`.

Table 1–6: Class and custom files for SmartB2BObjects

Class files		Customer files	
Definition	<code>b2b.cld</code>	Super	<code>b2bcustom.p</code>
Method	<code>b2b.i</code>	Method	<code>b2bcustom.i</code>
Property	<code>b2bprop.i</code>	Property	<code>b2bpropcustom.i</code>
Prototype	<code>b2bprto.i</code>	Prototype	<code>b2bprtocustom.i</code>
Template	<code>b2b.w</code> <code>dynb2b.w</code>	Exclude	<code>b2bexclcustom.i</code>
–	–	Instance	<code>b2bdefscustom.i</code>

Examples: MsgHandler class

The MsgHandler class is represented by the super procedure `msghandler.p`. [Table 1–7](#) lists the class and custom files related to `msghandler.p`.

Table 1–7: Class and custom files for MsgHandler

Class files		Customer files	
Definition	<code>msghandler.cld</code>	Super	<code>msghandlercustom.p</code>
Method	<code>msghandler.i</code>	Method	<code>msghandlercustom.i</code>
Property	<code>msghprop.i</code>	Property	<code>msghpropcustom.i</code>
Prototype	<code>msghprto.i</code>	Prototype	<code>msghprtocustom.i</code>
Template	<code>msghandler.w</code>	Exclude	<code>msghandlerexclcustom.i</code>
–	–	Instance	<code>msghandlerdefscustom.i</code>

Examples: XML class

The `xm1.p` file presents a simplified DOM API by encapsulating the 4GL DOM statements so that `x-noderef` handles never need to be exposed to `b2b`. [Table 1–8](#) lists the class and custom files related to `xm1.p`.

Table 1–8: Class and custom files for XML

Class files		Customer files	
Definition	<code>xm1.cld</code>	Super	<code>xm1custom.p</code>
Method	<code>xm1.i</code>	Method	<code>xm1custom.i</code>
Property	<code>xm1prop.i</code>	Property	<code>xm1propcustom.i</code>
Prototype	<code>xm1prto.i</code>	Prototype	<code>xm1prtocustom.i</code>
Template	<code>xm1.w</code>	Exclude	<code>xm1exclcustom.i</code>
—	—	Instance	<code>xm1defscustom.i</code>

SmartCombo

The SmartCombo extends ADM2 Smart technology to the Combo Box level. The class `Combo` is a user-defined class. [Table 1–9](#) lists the class and custom files related to `combo.p`.

Table 1–9: Class and custom files for SmartCombo

Class files		Customer files	
Definition	<code>combo.cld</code>	Super	<code>combocustom.p</code>
Method	<code>combo.i</code>	Method	<code>combocustom.i</code>
Property	<code>combprop.i</code>	Property	<code>combopropcustom.i</code>
Prototype	<code>combprto.i</code>	Prototype	<code>comboprtocustom.i</code>
Template	<code>dyncombo.w</code>	Exclude	<code>comboexclcustom.i</code>
—	—	Instance	<code>combodefscustom.i</code>

SmartConsumer

See the “[SmartProducer and 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 `src/adm2/sbo.p` is its super procedure. Browser inherits from Progress class DataVis. [Table 1–10](#) lists the class and custom files related to browser.p.

Table 1–10: Class and custom files for SmartDataBrowser

Class files		Customer 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	brschng.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 `src/adm2/field.p` is its super procedure. [Table 1–11](#) lists the class and custom files related to `field.p`.

Table 1–11: Class and custom files for SmartDataField

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 runs 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 `src/adm2/data.p` is Data's super procedure. [Table 1–12](#) lists the class and custom files related to `data.p`.

Table 1–12: Custom and class files for SmartDataObjects

Class files		Custom files	
Definition	<code>data.cld</code>	Super	<code>datacustom.p</code>
Method	<code>data.i</code>	Method	<code>datacustom.i</code>
Property	<code>dataprop.i</code>	Property	<code>datapropcustom.i</code>
Prototype	<code>dataprto.i</code>	Prototype	<code>dataprtocustom.i</code>
Template	<code>data.w</code> <code>dyndata.w</code>	Exclude	<code>dataexclcustom.i</code>
Overflow Files	<code>dataext.p</code> <code>dataextcols.p</code>	Instance	<code>datadefscustom.i</code>
Additional Method Files	<code>cltorsvr.i</code> <code>robjflds.i</code>	—	—

Examples: Query class

The super-procedure file for class Query is `query.p`. [Table 1–13](#) lists the class and custom files related to `query.p`.

Table 1–13: Custom and class files for queries

(1 of 2)

Class files		Custom files	
Definition	<code>query.cld</code>	Super	<code>querycustom.p</code>
Method	<code>query.i</code>	Method	<code>querycustom.i</code>
Property	<code>qryprop.i</code>	Property	<code>qrypropcustom.i</code>
Prototype	<code>qryprto.i</code>	Prototype	<code>qryprtocustom.i</code>
Template	—	Exclude	<code>queryexclcustom.i</code>

Table 1–13: Custom and class files for queries

(2 of 2)

Class files		Custom files	
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 inherits 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 src/adm2/data.p is its super procedure.

Table 1–14 lists the class and custom files related to viewer.p.

Table 1–14: Class and custom files for SmartDataViewer

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 `src/adm2/container.p` is its super procedure. [Table 1–15](#) lists the class and custom files related to `container.p` and the SmartDialog.

Table 1–15: Class and custom files for SmartDialog

Class files		Custom files	
Definition	–	Super	<code>containrcustom.p</code>
Method	<code>containr.i</code>	Method	<code>containrcustom.i</code>
Property	<code>cntnprop.i</code>	Property	<code>cntnpropcustom.i</code>
Prototype	<code>cntnprto.i</code>	Prototype	<code>cntnprtocustom.i</code>
Template	<code>cntnrdlg.w</code>	Exclude	<code>containrexclcustom.i</code>
Additional method file	<code>dialogmn.i</code>	Instance	<code>containrdefscustom.i</code>

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 `src/adm2/filter.p`. [Table 1–16](#) lists the class and custom files related to `filter.p`.

Table 1–16: Class and custom files for SmartFilter

Class files		Custom files	
Definition	<code>filter.cld</code>	Super	<code>filtercustom.p</code>
Method	<code>filter.i</code>	Method	<code>filtercustom.i</code>
Property	<code>filtprop.i</code>	Property	<code>filtpropcustom.i</code>
Prototype	<code>filtprto.i</code>	Prototype	<code>filtprtocustom.i</code>
Template	<code>dynfilter.w</code>	Exclude	<code>filterexclcustom.i</code>
–	–	Instance	<code>filterdefscustom.i</code>

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 `src/adm2/visual.p`. [Table 1–17](#) lists the class and custom files related to `visual.p`.

Table 1–17: Class and custom files for SmartFolder

Class files		Custom files	
Definition	–	Super	–
Method	–	Method	–
Property	–	Property	–
Prototype	–	Prototype	–
Template	<code>folder.w</code>	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 `src/adm2/container.p`. [Table 1–18](#) lists the class and custom files related to `container.p` and SmartFrame.

Table 1–18: Class and custom files for SmartFrame

Class files		Custom files	
Definition	–	Super	–
Method	–	Method	–
Property	–	Property	–
Prototype	–	Prototype	–
Template	<code>cntnrfrm.w</code>	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.

[Table 1–19](#) lists the class and custom files related to `lookup.p`.

Table 1–19: Class and custom files for SmartLookup

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 `src/adm2/panel.p`. [Table 1–20](#) lists the class and custom files related to `panel.p`.

Table 1–20: Class and custom files for SmartPanel (1 of 2)

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

Table 1–20: Class and custom files for SmartPanel (2 of 2)

Class files		Custom files	
Template	pcommit.w pnavico.w pnavlbl.w pupdsav.w	Exclude	panelexclcustom.i
—	—	Instance	paneldefscustom.i

SmartProducer and 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 `src/adm2/producer.p` is the super procedure for the class SmartProducer. The file `src/adm2/consumer.p` is the super procedure for the class SmartConsumer. [Table 1–21](#) lists the class and custom files related to `producer.p`.

Table 1–21: Class and custom files for SmartProducer

Class files		Custom files	
Definition	<code>producer.cld</code>	Super	<code>producercustom.p</code>
Method	<code>producer.i</code>	Method	<code>producercustom.i</code>
Property	<code>prodprop.i</code>	Property	<code>prodpropcustom.i</code>
Prototype	<code>prodprto.i</code>	Prototype	<code>prodprtocustom.i</code>
Template	<code>producer.w</code> <code>dynproducer.w</code>	Exclude	<code>producerexclcustom.i</code>
—	—	Instance	<code>producerdefscustom.i</code>

[Table 1–22](#) lists the files for the Consumer class.

Table 1–22: Class and custom files for Consumer

Class files		Custom files	
Definition	<code>consumer.cld</code>	Super	<code>consumercustom.p</code>
Method	<code>consumer.i</code>	Method	<code>consumercustom.i</code>
Property	<code>consprop.i</code>	Property	<code>conspropcustom.i</code>
Prototype	<code>consprto.i</code>	Prototype	<code>consprtocustom.i</code>
Template	<code>consumer.w</code> <code>dynconsumer.w</code>	Exclude	<code>consumerexclcustom.i</code>
—	—	Instance	<code>consumerdefscustom.i</code>

Messaging class

The file `src/adm2/messaging.p` is the super procedure for the class `Messaging`. [Table 1–23](#) lists the class and custom files related to `messaging.p`.

Table 1–23: Class and custom files for messaging

Class files		Custom files	
Definition	<code>messaging.cld</code>	Super	<code>messagingcustom.p</code>
Method	<code>messaging.i</code>	Method	<code>messagingcustom.i</code>
Property	<code>messprop.i</code>	Property	<code>messpropcustom.i</code>
Prototype	<code>messprto.i</code>	Prototype	<code>messprtocustom.i</code>
Template	—	Exclude	<code>messagingexclcustom.i</code>
—	—	Instance	<code>messagingdefscustom.i</code>

SmartRouter

The `SmartRouter` is a utility object that routes incoming documents to the appropriate `SmartB2BObject` for transformation. The `SmartRouter` uses the `Router` class, a user-defined class whose super-procedure file is `src/adm2/router.p`. [Table 1–24](#) lists the class and custom files related to `router.p`.

Table 1–24: Class and custom files for SmartRouter

Class files		Custom files	
Definition	<code>router.cld</code>	Super	<code>routercustom.p</code>
Method	<code>router.i</code>	Method	<code>routercustom.i</code>
Property	<code>routprop.i</code>	Property	<code>routpropcustom.i</code>
Prototype	<code>routprto.i</code>	Prototype	<code>routprtocustom.i</code>
Template	<code>router.w</code>	Exclude	<code>routerexclcustom.i</code>
—	—	Instance	<code>routerdefscustom.i</code>

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 src/adm2/select.p. [Table 1–25](#) lists the class and custom files related to select.p.

Table 1–25: Class and custom files for SmartSelect

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 and 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. [Table 1–26](#) lists the class and custom files related to msghandler.p for SmartSender and SmartReceiver.

Table 1–26: Class and custom files for SmartSender and SmartReceiver

(1 of 2)

Class files		Custom files	
Definition	msghandler.cld	Super	msghandlercustom.p
Method	msghandler.i	Method	msghandlercustom.i
Property	msghprop.i	Property	msghpropcustom.i

Table 1–26: Class and custom files for SmartSender and SmartReceiver*(2 of 2)*

Class files		Custom files	
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 toolbar. The default SmartToolbar can replace all the dedicated SmartPanels that are also distributed with AppBuilder. The SmartToolbar object also provides access to toolbar and menu actions that you want to define.

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. The Toolbar super-procedure file is `src/adm2/toolbar.p`. [Table 1–27](#) lists the class and custom files related to `toolbar.p`.

Table 1–27: Class and custom files for SmartToolbar*(1 of 2)*

Class files		Custom files	
Definition	toolbar.cld	Super	toolbarcustom.p
Method	toolbar.i	Method	toolbarcustom.i actioncustom.i
Property	toolprop.i	Property	toolpropcustom.i actiprtocustom.i
Prototype	toolprto.i	Prototype	toolprtocustom.i actiprtocustom.i

Table 1–27: Class and custom files for SmartToolbar (2 of 2)

Class files		Custom files	
Template	toolbar.w dyntoolbar.w	Exclude	toolbarexclcustom.i actionexclcustom.i
Overflow Files	toolbarext.p	Instance	toolbardefscustom.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 src/adm2/container.p. [Table 1–28](#) lists the class and custom files related to container.p for SmartWindow.

Table 1–28: Class and custom files for SmartWindow

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 `ADMPProps` temp-table. That table and its global handle `ghADMPProps` 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 `ADMPProps` 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`.

Refer to [Figure 1–1](#) to see the inheritance hierarchy for each object class.

***getpropname* and *setpropname* 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 *getpropname*. 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 *setpropname*. 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 1–2 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 1–2: Commented source code of GET pseudo-function

SmartObjects and Their Methods and Properties

This chapter lists and describes the methods (internal procedures and functions) and properties used for the base ADM2 SmartObject. The base SmartObject is used for defining all other objects. Refer to [Figure 1–1](#) to see the inheritance hierarchy for each object class.

NOTE: For information specific to the WebSpeed environment (see the [“Alphabetical Listing Of WebSpeed-specific API Routines”](#) chapter).

Base methods for SmartObjects

The following section describes the base methods and general super procedures for the SmartObjects used to create ADM2 applications.

addLink

Procedure that adds a link between two objects by setting property values in each.

Location: smart.p

Parameters:

INPUT phSource AS HANDLE

Source procedure handle.

INPUT pcLink AS CHARACTER

Link name.

INPUT phTarget AS HANDLE

Target procedure handle.

Notes:

- A NavigationSource (toolbar) linked to a SmartBusinessObject navigates the SmartDataObject that is linked to the visual DataTarget on a paged container.
- 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 is 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 ADM2 or Progress AppBuilder for more information about PassThrough links.

- If the link name is of the type **PageN**, 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 adds 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 defines 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

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

Location: smart.p

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.

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 are not seen by the Translation Manager tool.
- If the unknown value (?) is passed as the message text value, then addMessage retrieves 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, ?).

```

adjustTabOrder

Procedure that changes the tab order of SmartObjects.

Location: smart.p

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 anchors the SmartObject.

INPUT pcPosition AS CHARACTER

“After” if the SmartObject is moved after the anchor.

“Before” if the SmartObject is moved before the anchor.

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 makes 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).

```

anyMessage

Returns a flag indicating whether there are messages in the error message log.

Location: smart.p

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 messages are 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 */
```

applyEntry

Procedure that 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.

Location: smart.p

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.

Notes: None

assignLinkProperty

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

Location: smart.p

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).
```

```
/* 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).
```

changeCursor

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

Location: smart.p

Parameters:

INPUT pcCursor AS CHARACTER

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

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.

createControls

Procedure that defines the default action for SmartObject-specific initialization of ActiveX Controls. Runs adm-create-controls, an AppBuilder-generated procedure.

Location: smart.p

Parameters: None

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.

destroyObject

Procedure that cleans up and deletes the current object procedure and its descendents, if any.

Location: `smart.p`

Parameters: None

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 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 PUBLISH destroyObject to delete all the SmartObject procedures they contain, before destroying themselves.
- You can localize the destroyObject procedure to add a check to do cleanup before a destroy completes, or to stop a destroy event from finishing. (See the [confirmExit](#) entry for the standard event procedure for doing this.)

displayLinks

Utility procedure used to put up a dialog showing all the ADM links for a given container object.

Location: `smart.p`

Parameters: None

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.

editInstanceProperties

Procedure that runs the dialog to get run-time property settings.

Location: `smart.p`

Parameters: None

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 is 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.

```

exitObject

Procedure that passes an exit request to its container.

Location: smart.p

Parameters: None

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 (for example, when a 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
```

fetchMessages

Returns a delimited list of all messages in their raw form. The message log is cleared.

Location: smart.p

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).
- The fetchMessages function clears the message log, and it is the responsibility of the calling procedure to display the messages or handle them appropriately. Use the similar function reviewMessages to read messages without deleting them.

fixQueryString

Conventionalizes decimal delimiters in query strings to be full-stops.

Location: smart.p

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.

initializeObject

Procedure that performs general initialization common to all objects.

Location: smart.p

Parameters: None

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 runs initializeObject. This passes the initializeObject event down through all the contained objects. Therefore, any version of initializeObject can assume that all the SmartObjects and SmartLinks in a container have been established and that they can be checked, for example, whether there is a link to some other particular kind of object. Code should not, however, assume the order in which SmartObjects are initialized.

For example, a SmartDataObject opens its query and publishes **dataAvailable** to signal that to other objects. A SmartDataViewer runs 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.

instanceOf

Takes an object type as input and resolves the query inheritance as defined in the Dynamics Repository.

Location: smart.p

Parameters:

INPUT pcObjectType AS CHARACTER

The object type for which you want to resolve the inheritance.

Notes:

- This function is used only for a Progress Dynamics environment.
- Returns TRUE if the instance inherits from the requested class anywhere in the hierarchy.

instancePropertyList

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.

Location: smart.p

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.

hideObject

Procedure that hides the current object if it is a visual object and sets the `ObjectHidden` property to `TRUE` to indicate the state of the object.

Location: `smart.p`

Parameters: None

Notes:

- The `Hide` concept is a logical one; 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` does 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 are hidden along with the container. For this reason, `hideObject`, when executed for a `SmartContainer`, sets the property `ContainerHidden` in each contained `SmartObject` without actually running `hideObject` in each object. The `setContainerHidden` function, in turn, sets 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.

```

linkHandles

Takes a link name and returns a list of handles of objects at the other end of that link, relative to the TARGET-PROCEDURE.

Location: smart.p

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

Returns the requested property in the object at the other end of the specified link, relative to TARGET-PROCEDURE.

Location: smart.p

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).
```

linkStateHandler

Procedure Handler for the linkState event. This procedure is also used by addLink and removeLink to subscribe and unsubscribe to the link events in the object.

Location: smart.p

Parameters:

INPUT pcState AS CHARACTER

Mode for the object. The valid values are:

- **Add** — Activate new link by subscribing to the link events of the passed object.
- **Remove** — Deactivate removed link by unsubscribing to the link events of the passed object.
- **Active** — Activate links by subscribing to the link events of the passed object.
- **Inactive** — Deactivate links by unsubscribing to the link events of the passed object.

INPUT phObject AS HANDLE

Object to which you want to subscribe or unsubscribe.

INPUT pcLink AS CHARACTER

Full link name pointing to the passed object. Both **DataSource** and **Data-source** are supported.

NOTE: The name handler attempts to indicate that this is an event handler that should not be called directly outside of the intended events, but instead be actively used as an event to ensure that properties that are link dependant are set for removal.

mappedEntry

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.

Location: smart.p

Parameters:

INPUT pcEntry AS CHARACTER

Entry to lookup.

INPUT pcList AS CHARACTER

Comma-separated list with paired entries.

INPUT p1First 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.

messageNumber

Returns the message text given a message number. Allows these messages to be translated and tracked in one place.

Location: smart.p

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/admmmsgs.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 admmmsgs.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), ?, ?).
```

modifyListProperty

Procedure that allows values to be added to or deleted from any object property that is a comma-separated list.

Location: smart.p

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.

Notes:

- This is the ADM 2 equivalent of what was modify-list-attribute in the Version 8 ADM.
- Normally the first argument is 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 *getpropname* 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 *setpropname* function is run to reset the value of the list property. Both the *get* and *set* functions must exist; otherwise, modifyListProperty returns 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 *setpropname* function resets 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 specified Target procedure
   handle (from addLink). */

RUN modifyListProperty (INPUT phTarget, INPUT 'ADD':U, INPUT
   pcLink + "Source":U, INPUT STRING(phSource)).

```

modifyUserLinks

Procedure that 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.

Location: smart.p

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.

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 preserves compatibility with any possible changes.

oneObjectLinks

Procedure that adds linkage for some object.

Location: smart.p

Parameters:

INPUT hObject AS HANDLE

Handle of the object to be linked.

Notes: Called by displayLinks.

propertyType

Locates the **set** property function for the specified property name either locally or in a SUPER procedure, and returns its data type.

Location: smart.p

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 ° ) /* To add the string to a list. */
ELSE dynamic-function('set':U + pcLinkType + "Source":U, INPUT phTarget).
```

removeAllLinks

Procedure that removes all links for a SmartObject, normally as part of destroying a SmartObject procedure.

Location: smart.p

Parameters: None

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).

removeLink

Procedure that removes a specific link between two objects.

Location: smart.p

Parameters:

INPUT phSource AS HANDLE

Source procedure handle.

INPUT pcLink AS CHARACTER

Link type name.

INPUT phTarget AS HANDLE

Link target object handle.

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).
```

repositionObject

Procedure that adjusts the position of container objects.

Location: smart.p

Parameters:

INPUT pdRow AS DECIMAL

INPUT pdCol AS DECIMAL

Notes: None

returnFocus

Procedure that returns focus to the containing window.

Location: smart.p

Parameters:

INPUT hTarget AS HANDLE

A handle to the target procedure object.

Notes: None

reviewMessages

Returns a delimited list of all messages without removing them from the log.

Location: smart.p

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.

showMessage

Displays, using a simple MESSAGE statement by default, either a literal message string or the return value from a call to messageNumber.

Location: smart.p

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

Procedure used by Progress Dynamics override for showMessage function to use Progress 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.

Location: smart.p

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 p1Answer AS LOGICAL

The user's choice of button: TRUE or FALSE (**Question** or **YesNo**), or TRUE/FALSE/UNDEFINED (**YesNoCancel**)

Notes: None

Examples:

```
showMessageProcedure ("Is this not a nice &1 &2 message?", "short", "sample",  
"Question").
```

Signature

Returns the signature of the named function or internal procedure in the format returned by the Progress GET-SIGNATURE method.

Location: smart.p

Parameters:

INPUT pcName AS CHARACTER

The function or procedure name.

Returns: CHARACTER (Signature in Progress GET-SIGNATURE format.)

Notes: None

start-super-proc

Procedure that starts a super procedure if it is not already running and adds it as a super procedure in any case.

Location: smart.i

Parameters:

INPUT pcProcName AS CHARACTER

Notes: None

viewObject

Procedure that logically views the current object and sets its ObjectHidden property to FALSE.

Location: smart.p

Parameters: None

Notes:

- The ADM supports a logical concept of viewing that you can apply to all objects regardless of whether they have a visualization.

- When an object is:
 - Viewed, the linkState property is typically set to **active** which activates the links.
 - Hidden, the link is sometimes deactivated, depending on the link type. If an object has an actual visualization, the version of viewObject in `visual.p` views 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, sets the property `ContainerHidden` to `FALSE` in each contained SmartObject, without actually running `viewObject` in each object. The `setContainerHidden` function then sets the `ObjectHidden` property to `FALSE` 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: Example

```
/* Views an object when a button is pressed. */  
  
ON CHOOSE OF Btn_View_Browser  
DO:  
    RUN viewObject IN hSDBrowser.  
END.
```

AppServer methods for SmartObjects

The AppServer methods expend the Container and Query classes. If you define the macro `{&APP-SERVER-VARS}`, the Container and Query classes inherit from the AppServer class. the following section describes the AppServer methods.

bindServer

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

Location: `appserver.p`

Parameters: None

Notes: None

destroyObject

Procedure that disconnects the AppServer connection if present before invoking the standard destroy code.

Location: appserver.p

Parameters: None

Notes: None

destroyServerObject

Destroys the server object and retrieves its context.

Location: appserver.p

Parameters: None

Notes: INPUT

- Called from unbindServer when stateless.
- This event is not always called because queryObjects now supports singlet data requests or calls destroyObject directly when BindScope is data.

disconnectObject

Procedure that disables fields in the ENABLED-FIELDS list.

Location: appserver.p

Parameters: None

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.

initializeServerObject

Procedure that initializes the server object after it has been started or restarted

Location: appserver.p

Parameters: None

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.

runServerObject

Procedure that runs the server part of this object and sets AShandle.

Location: appserver.p

Parameters:

INPUT phAppService AS HANDLE

AppServer Session handle.

Notes: Called from startServerObject and restartServerObject.

runServerProcedure

Returns the handle of a procedure after it runs on server.

Location: appserver.p

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.

unbindServer

Procedure that unbinds the AppServer by destroying the server side object started by the client.

Location: appserver.p

Parameters:

INPUT pcMode AS CHARACTER

Valid values are **unconditional**, **conditional**, and the empty string. If **conditional**, unbinding only takes place if the caller is at the same level as BindSignature. The empty string is recognized, but not currently supported.

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 uses 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 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).

SmartObject properties

SmartObject properties provide information about SmartObject and their classes. This information can include whether an object is enabled, the contents of the object and so on. You can read property values and in many instances you can change property values. To read a value for a property, you use a **get** function, and to change a value for a property, you use a **set** function.

These functions conform to the following conventions:

- **get** — Uses the form *getpropnam* and returns the current value of the property

NOTE: This function accepts no arguments.

- **set** — Uses the form *setpropname*. The set function accepts a single argument—the new value for the property—and returns TRUE/FALSE depending on whether the value change succeeds.

For more information about getting and setting property values, see [Chapter 1, “ADM2 SmartObject API Reference”](#) in this guide.

The following section lists and describes the SmartObject properties that you can use with the get and set functions. The description also identifies the properties for which you can read and write (change) a value and the properties for which you can only read the value.

AppService

Logical partition name of the stored AppService used to connect to an AppServer. This value identifies the AppService in which the SmartDataObject is to be started.

Data Type: CHARACTER

Notes: Read and Write

ASBound

Indicates whether this object binds the AppServer with a persistently running procedure: usually the server part of this object. If the object binds the AppServer with a persistently running procedure, the value for this property is TRUE.

Data Type: LOGICAL

Notes: Read only

AsDivision

A string indicating whether the object is running on the client side or the server side of the AppServer. If this value is an empty string, there is no AppServer.

Data Type: CHARACTER

Notes: Read and Write

ASHandle

Handle of the persistent procedure to this object's companion procedure (the copy of itself) running on the AppServer.

Data Type: HANDLE

Notes: Read and Write

ASHasStarted

Indicates where the object has completed its first call to its server-side object. This value is TRUE if the object has completed its first call.

Data Type: LOGICAL

Notes: Read only

ASInitializeOnRun

Indicates whether runServerObject should call initializeServerObject. initializeServerObject is called on the client, but usually has a call to the server for context. This value is TRUE if runServerObject should call initializeServerObject.

Data Type: LOGICAL

Notes: Read and Write

ChildDataKey

When a program is run for a specific record, that is a specific customer, ChildDataKey would record that this instance of the object is for this specific piece of child data, and the key would represent the key passed into the object.

Then, when selecting a record in an object controller, you can use this key to determine if an object containing data for the selected key is already open. If that is the case, that window can be brought to the top.

Data Type: CHARACTER

Notes:

- Read and Write
- Used in a Progress Dynamics environment by the Session Manager when it is keeping track of the running instance of programs.

- Use this property when multiple windows are supported, and you do not want to bring up two windows containing the same child data.
- ChildDataKey is an input parameter to the launch container api.

ContainerHandle

Though available for all SmartObjects, this property is most meaningful for SmartObjects that have a window or frame. For a frame-based object, such as a Browser or Viewer, this property returns frame. For a window-based objects such as a SmartWindow, this property returns the window handle. You can use this information when you need to query or manipulate the properties of the container widget itself.

Data Type: HANDLE

Notes: Read only

ContainerHidden

Indicates whether or not an object's SmartContainer: SmartWindow, SmartFrame and so on, is hidden. Returns TRUE if the container is hidden.

Data Type: LOGICAL

Notes: Read and Write

ContainerSource

Handle of the object that should become the Container-Source.

Data Type: HANDLE

Notes: Read and Write

ContainerSourceEvents

Comma-separated list of the events to which this object wants to subscribe to in its ContainerSource.

Data Type: CHARACTER

Notes: Read and Write

ContainerType

Type of container this SmartObject is (**Window** or **Frame**), or the empty string if the object is not a container.

Data Type: CHARACTER

Notes: Read only

DataLinksEnabled

Indicates whether or not a data links are enabled.

Data Type: LOGICAL

Notes: Read and Write

DataSource

Object's data source, if any.

Data Type: HANDLE

Notes: Read and Write

DataSourceEvents

Comma-separated list of the events this object wants to subscribe to in its data source.

Data Type: CHARACTER

Notes: Read and Write

DataSourceNames

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.

Data Type: CHARACTER

Notes: Read and Write

DataTarget

Handle in character format, or a comma-separated list of handles for the case of multiple data targets.

Data Type: CHARACTER

Notes: Read and Write

DataTargetEvents

List of events this object class should be subscribed to in its data targets.

Data Type: CHARACTER

Notes: Read and Write

DBAware

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.

Data Type: LOGICAL

Notes: Read and Write

inactiveLinks

Comma-separated paired list of inactive links. The second entry of each pair is a semicolon-separated list of object handles.

Data Type: Logical

Notes:

- Read and Write
- `modifyInactiveLinks` should be used to maintain this property.
- `isLinkInactive` should be used to check if an actual link is inactive.

InstanceProperties

A list of the ADM instance properties of the SmartObject. Instance properties are those properties that can be set at design time for initialization as part of startup.

Data Type: CHARACTER

Notes: Read and Write

LogicalObjectName

Value of LogicalObjectName.

Data Type: CHARACTER

Notes: Read and Write

LogicalVersion

Version number of the object name as stored in the Dynamics Repository. This value is read from the rym_data_version table which keeps a record of all repository object versions for version managed data, for example, ryc_smartobject objects.

Data Type: CHARACTER

Notes: Read and Write

ObjectHidden

Flag indicating whether or not 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.

Data Type: LOGICAL

Notes: Read and Write

ObjectInitialized

Flag indicating whether this object has been initialized. TRUE if this object has been initialized.

Data Type: LOGICAL

Notes: Read only

ObjectName

Name of the object that can be the filename or some other designation meaningful to the repository and other objects.

Data Type: CHARACTER

Notes: Read and Write

ObjectPage

Logical page on which this object has been placed.

Data Type: INTEGER

Notes: Read only

ObjectParent

Widget handle of this object's Window or Frame parent, that is, the handle of the visual container of its CONTAINER-SOURCE.

Data Type: HANDLE

Notes: Read and Write

ObjectType

Type of the SmartObject, such as SmartDataObject and so on. If this is a super procedure, **SUPER** is returned.

Data Type: CHARACTER

Notes: Read only

ObjectVersion

ADM version of the SmartObject.

Data Type: CHARACTER

Notes: Read and Write

ParentDataKey

ParentData Key.

Data Type: CHARACTER

Notes: Read and Write

PassThroughLinks

List of link types that can be pass-through links. Used by addLink. This property is shared by all objects should normally be modified using modifyListProperty.

Data Type: CHARACTER

Notes: Read and Write

PhysicalObjectName

The name of the physical file run to instantiate either a static or Dynamics object.

Static objects have only a physical object name. Dynamics objects have a logical object name which is the name of the repository object, and a physical object name which is the name of the rendering object used to build the Dynamics object. For example, a dynamic browser could have a logical object name of **customerbrowse** and a physical object name of **rydynbrowb.w**.

Data Type: CHARACTER

Notes:

- Read and Write
- This property includes the file extension but does not include the path for the file name.
- This property is set internally and should not be changed by application code.

PhysicalVersion

Version number of the physical object. This number is displayed in the Help About message dialog box. The version number for all Dynamics objects are created using the standard templates.

If Roundtable is being used, this version number is automatically updated to match the version number that is in Roundtable for the object. If Roundtable is not being used, you must manually update this version number.

Data Type: CHARACTER

Notes: Read and Write

PropertyDialog

Name of the dialog procedure that sets InstanceProperties.

Data Type: CHARACTER

Notes: Read only

QueryObject

Flag that indicates whether this object manages its own database query.

Data Type: LOGICAL

Notes: Read only

RunAttribute

Property that maps to attributes set up in the gsc_instance_attribute table in the Dynamics Repository. Use this property to change the behavior of generic objects. For example, If you have a generic object that you want to behave differently for a creditors system and a debtors system. you can use this property to determine the functionality for the specific instance of the generic object. That is, is the object being used in the creditors system or the debtors system.

Data Type: CHARACTER

Notes:

- Read and Write
- The mapped value is used for the container and all contained objects.
- The property is posted to the program using either the menu option the program was run from or coded in a program if the program was run from a button and so on. As a result, some of these properties are owned by the system and cannot be maintained or deleted by users.
- When security is applied, for example field-level security, they can be defined globally, for a specific product, product module or to an individual program level. The RunAttribute property is a level below the program level that permits security settings per instance of a program

ServerFileName

Property representing the actual server-side SDO filename to run on the AppServer. This value might not be the ObjectName if that has been modified.

Data Type: CHARACTER

Notes: Read and Write

ServerOperatingMode

A string representing the connection state. Valid values are **stateless**, **state-reset**, **state-aware**, or **none** (no AppServer connection).

Data Type: CHARACTER

Notes: Read and Write

SupportedLinks

Comma-separated list of the SmartObject links supported by this object.

Data Type: CHARACTER

Notes: Read and Write

TranslatableProperties

List of properties that should not have a :U following their literal values when code is generated in adm-create-objects. Because this is a comma-separated list, it should normally be invoked indirectly through modifyListAttribute.

Data Type: CHARACTER

Notes: Read and Write

UIBMode

Indicates whether an object is in Design mode in AppBuilder (AppBuilder was originally called User-interface Builder). This value is blank if the object is not in design mode, that is, not running in an AppBuilder design window. This value is **Design** if the object is in design mode, or **Design-Child** if it is contained in another SmartObject that is in design mode, such as a SmartFrame. This value is ? if the object is not a SmartObject and does not have a valid handle in ADM-DATA.

Data Type: CHARACTER

Notes: Read and Write

UseRepository

Indicates whether the repository is running and available. If this value is TRUE, then Progress Dynamics is running and the Repository is available.

Data Type: LOGICAL

Notes: Read only

UserProperty

Dynamically defined property.

Data Type:

Notes:

- Write only
- You cannot set or get user-defined properties in an application running in n-tier (client and AppServer) mode.

Visual Objects and Their Methods and Properties

This chapter lists and describes the methods (internal procedures and functions) and properties for visual objects. Refer to [Figure 1–1](#) to see the inheritance hierarchy for each object class.

For information specific to the WebSpeed environment (see the “[Alphabetical Listing Of WebSpeed-specific API Routines](#)” chapter).

Base methods for visual objects

The following section describes the base methods for visual objects.

applyLayout

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

Location: visual.p

Parameters: None

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.

assignFocusedWidget

Sets focus to the named object. Not supported for SmartDataFields and returns FALSE if attempted for SmartDataFields.

Location: visual.p

Parameters:

INPUT name AS CHARACTER

The name of a single Dynamics object.

Returns: LOGICAL

Notes: None

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF widgetIsModified("name":U) THEN  
  assignFocusedWidget("description":U).
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
if (logic.widgetIsModified('orderviewv.name'))
    logic.assignFocusedWidget('orderviewv.description');
```

assignWidgetValue

Takes the name of one object and a character screen value as input and sets the SCREEN-VALUE of the object. The DataValue is set for a SmartDataField. (This would always be the key field for a lookup even when the display field is different.)

Location: visual.p

Parameters:

INPUT name AS CHARACTER

The name of a single Dynamics object.

INPUT value AS CHARACTER

A string representing the intended screen value of the widget.

Returns: LOGICAL

Notes:

- Sets DataModified to make the toolbar enable saving data (it behaves as if the user actually changed the field value manually).
- Sets DataModified to TRUE whether or not the field is enabled.

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
assignWidgetValue("Browse.Status":U, "Shipped":U).
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
logic.assignWidgetValue('orderfullo.browse.status','Shipped');
```

assignWidgetValueList

Takes the name of one or more objects and character screen values and a delimiter as input and sets the SCREEN-VALUE of the objects. The DataValue is set for a SmartDataField (this would always be the key field for a lookup even where the display field is different).

This function sets the DataModified attribute to force the toolbar to enable saving data (it behaves as if the user actually changed the field value manually). The DataModified attribute is set to TRUE whether the field is enabled or not.

Location: visual.p

Parameters:

INPUT namelist AS CHARACTER

The name of one or more objects, separated by commas.

INPUT valuelist AS CHARACTER

A string representing the intended screen values for one or more widgets separated by the specified or default delimiter.

If the name list contains more than one object, then the value list must be either a single value, which will be applied to each field in the name list, or it must be a list of an equal number of entries to the name list, in which case the values are assigned to the corresponding object names. If there is more than one entry in the value list and the number of entries in the namelist and the number of entries in the value list do not match, nothing is done and FALSE is returned. If any assignments fail because the widget is not found, FALSE is returned but it still processes through the list and assigns those that it can.

INPUT delimiter AS CHARACTER

(Optional) Specifies the delimiter to use for the value list. Specify “?” to use the default delimiter, which is the pipe character (“|”).

Returns: LOGICAL

Notes: For the DHTML client, the delimiter is optional as the third parameter. If it is not specified, the function uses the default, which is the pipe symbol (“|”).

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
assignWidgetValueList("city,state":U, "Boston|MA":U, ?).
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
1Ret=logic.assignWidgetValueList('customerviewv.name,customerviewv.city',
'Reeds Ferry|Nashua');
```

blankWidget

Blanks the SCREEN-VALUE of the objects in the namelist. The DataValue is blanked for a SmartDataField (this would always be the key field for a lookup even where the display field is different).

Location: visual.p

Parameters:

INPUT namelist AS CHARACTER

The name of one or more objects, separated by commas.

Returns: LOGICAL

Notes:

- This function sets the DataModified attribute to force the toolbar to enable saving data (it behaves as if the user actually changed the field value manually).
- This function does nothing to objects that do not support SCREEN-VALUE and to objects where a blank screen value does not make sense, such as toggle boxes. It blanks a combo-box by setting its list items to its list items.

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF widgetIsTrue("customerviewv.disableAccount":U) THEN
blankWidget("Self.accountID":U).
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
if (logic.widgetIsTrue('customerviewv.disableAccount'))
logic.blankWidget('myviewv.accountID');
```

disableObject

Procedure that disables all enabled objects in a frame, including RowObject fields.

Location: visual.p

Parameters: None

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 does 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.

disableRadioButton

Disables the specified radio button of the radio set objects identified in the namelist. Returns FALSE if a widget in the list is not a radio-set, if a widget in the list is not found, or if the button number is invalid.

Location: visual.p

Parameters:

INPUT namelist AS CHARACTER

The name of one or more radio-set objects, separated by commas.

INPUT buttonNum AS INTEGER

The numerical ID of a radio button in a particular radio set.

Returns: LOGICAL

Notes: This API is not supported by the DHTML client.

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF widgetIsTrue("shipped":U) THEN
  disableRadioButton("shippingTrans":U, 2).
```


disableWidget

Disables the object or objects identified in the namelist.

For SmartDataFields, it runs the disableField function. disableField is not a generic SmartDataField function, therefore this API is only supported for SmartDataFields that have disableField defined. If a field in the list is not found, or a SmartDataField without disableField is included in the list, disableWidget returns FALSE.

Location: visual.p

Parameters:

INPUT namelist AS CHARACTER

The name of one or more objects, separated by commas.

Returns: LOGICAL

Notes:

- Data fields are removed from the EnabledFields and EnabledHandles properties to ensure that once the field is disabled by the API, it remains disabled,
- Local fields are removed from the EnabledObjFldsToDisable property when the property is set to a field list. When the EnabledObjFldsToDisable property is set to **(All)**, the disableWidget function sets this property to the value of the EnabledObjFlds property without the field being disabled. This action ensures that once a field is disabled by the API, it remains disabled. When the EnabledObjFldsToDisable property is set to **(None)**, it is not changed.

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF widgetValue("Browse.Terms":U) NE "" :U THEN
  disableWidget("Self.Discount":U).
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
if (logic.widgetValue('orderfullo.browse.terms')!='')
  logic.disableWidget('customervieww.discount');
```

enableObject

Procedure that enables an object. Includes all components except RowObject fields. They are enabled using EnableFields().

Location: visual.p

Parameters: None

Notes:

- enableObject is invoked from initializeObject to enable a SmartObject (different from when a SmartObject's RowObject fields are enabled with enableFields).
- You can customize enableObject when a SmartObject is enabled, perhaps for initialization that is needed each time a SmartObject is enabled, or to enable other related SmartObjects.
- 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.

enableRadioButton

Enables the specified radio button of the radio set objects identified in the name list. Returns FALSE if a widget in the list is not a radio-set, if a widget in the list is not found, or if the button number is invalid.

Location: visual.p

Parameters:

INPUT namelist AS CHARACTER

The name of one or more radio-set objects, separated by commas.

INPUT buttonNum AS INTEGER

The numerical ID of a radio button in a particular radio set.

Returns: LOGICAL

Notes: This API is not supported by the DHTML client.

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF NOT widgetIsTrue("shipped":U) THEN
  enableRadioButton("shippingTrans":U, 2).
```

enableWidget

Enables the object or objects identified in the name list. For SmartDataFields, it runs the enableField function. Note that enableField is not a generic SmartDataField function, therefore this API is only supported for SmartDataFields that have enableField defined. If a field in the list is not found, or a SmartDataField without enableField is included in the list, enableWidget returns FALSE.

Location: visual.p

Parameters:

INPUT namelist AS CHARACTER

The name of one or more objects, separated by commas.

Returns: LOGICAL

Notes:

- Data fields are added to the EnabledFields and EnabledHandles properties to ensure that once the field is enabled by the API, it remains so enabled.
- Local fields are added to the EnabledObjFldsToDisable property when the property is not set to (**None**) or (**All**). This action ensures that once a field is enabled by the API, it remains enabled. It could also result in adding a field that was not included in the EnabledObjFldsToDisable property initially to the property by a call to this API. If this behavior is not desired, do not use the enableWidget function or manipulate the EnabledObjFldsToDisable property after using the function to remove the field.

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF widgetIsModified("customervieww.discount":U) THEN
  enableWidget("Self.terms":U).
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
if (logic.widgetIsModified('customerviewv.discount'))  
    logic.enableWidget('orderviewv.terms');
```

formattedWidgetValue

Returns the SCREEN-VALUE of the object, or in the case of a browse column when in a ROW-DISPLAY trigger, the STRING-VALUE from the RowObject buffer field. The DataValue is returned for a SmartDataField (this would always be the key field for a lookup even where the display field is different). For example, you could use this function to get the formatted value of a single field to use for comparisons.

Location: visual.p

Parameters:

INPUT name AS CHARACTER

The name of a single Dynamics object.

Returns: CHARACTER

Notes: None

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF formattedWidgetValue("orderviewv.ordernum":U) > cNumber THEN ...
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
if (logic.formattedWidgetValue('orderviewv.ordernum')>cNumber) ...
```

formattedWidgetValueList

Takes the name of one or more objects and returns the SCREEN-VALUE of the object or objects. For example, use this function to retrieve the formatted values of several fields to assign their screen values to other fields.

If the object is a browse column (called from within a ROW-DISPLAY trigger), the STRING-VALUE from the RowObject buffer field is added to the list of returned values.

If the object is a SmartDataField, the DataValue field is returned. The DataValue field is always the key field, even when the display field is different.

If a field in the list is not found or a field has an unknown value, the function returns “?” for its value in the returned character list.

Location: visual.p

Parameters:

INPUT `namelist` AS CHARACTER

 The name of one or more objects, separated by commas.

INPUT `delimiter` AS CHARACTER

 (Optional) Specifies the delimiter to use for the return value only. Specify “?” to use the default delimiter, which is the pipe character (“|”).

Returns: CHARACTER

Notes: Use the pipe (“|”) delimiter in the DHTML client, since manipulating unprintable characters such as CHR(3) is problematic. The delimiter is optional. If not specified it defaults to the pipe delimiter.

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
ASSIGN
  cValues = formattedWidgetValueList("salesrepviewv.region,county,city":U,
  CHR(3))
  cCounty = ENTRY(2, cValues, CHR(3)).
  assignWidgetValue("county":U, cCounty).
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
cvalues=logic.formattedWidgetValueList('salesrepviewv.region,
salesrepviewv.city','|');
```

hideWidget

Hides the object or objects identified in the namelist (and their popup buttons). For SmartDataFields, it invokes the hideObject method.

Location: visual.p

Parameters:

INPUT namelist AS CHARACTER

The name of one or more objects, separated by commas.

Returns: LOGICAL

Notes: None

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF widgetIsTrue("Self.Current":U) THEN  
  hideWidget("Browse.Balance":U).
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
if (logic.widgetIsTrue('familyviewv.coveredonbenefits'))  
  logic.hideWidget('customerfullo.browse.balance');
```

highlightWidget

Sets the background and foreground colors (FGCOLOR and BGCOLOR widget attributes) of the named object or objects to a standard highlight color, depending on the value of the highlightType argument. For example, you may want to change the background color of a field with an invalid value to red.

Location: visual.p

Parameters:

INPUT namelist AS CHARACTER

The name of one or more objects, separated by commas.

INPUT highlightType AS CHARACTER

Determines the color values applied to the FGCOLOR and BGCOLOR widget attributes. The valid values of this parameter map to certain attributes of the visual class are:

Valid Value	Foreground Color	Background Color
info or information	colorInfoFG attribute	colorInfoBG attribute
warn or warning	colorWarnFG attribute	colorWarnBG attribute
err or error	colorErrorFG attribute	colorErrorBG attribute
def or default	Windows default	Windows default

The default setting uses the default Windows color values for foreground and background. The visual class attributes use values mapped to the application's color table. By default, the visual class attributes have the following values:

Attribute	Default color value
colorInfoFG	? (unknown value)
colorInfoBG	10 (green)
colorWarnFG	? (unknown value)
colorWarnBG	14 (yellow)
colorErrorFG	? (unknown value)
colorErrorBG	12 (red)

Returns: LOGICAL

Notes:

- To remove highlighting from a widget, call the function with the highlight type set to **default**.
- If code in one object, attempts to highlight a widget in another object, the color values come from the attribute the widget resides in.

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF widgetIsBlank("phone":U) THEN highlightWidget("phone":U, "warn":U).
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
logic.highlightWidget('customerviewv.phone,customerviewv.fax','warn');
```

initializeObject

Procedure that initializes code specific to visualizations.

Location: visual.p

Parameters: None

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.

resetWidgetValue

Resets the SCREEN-VALUE of the object or objects identified in the name list back to its original value from its data source. If a field in the list is not found or there is no data source for a field in the list, FALSE is returned and it will process all others in the list.

Location: visual.p

Parameters:

INPUT namelist AS CHARACTER

The name of one or more objects, separated by commas.

Returns: LOGICAL

Notes: None

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF widgetIsModified("itemId":U) THEN resetWidgetValue("discount":U).
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
if (logic.widgetIsModified('orderlinevieww.itemnum'))
    logic.resetWidgetValue('customervieww.discount');
```

toggleWidget

Reverses the value of one or more objects of type LOGICAL in the name list. The format of the widget is used to reverse its SCREEN-VALUE. For example, a logical with a format of credit/debit would change a “credit” screen-value to “debit.” A null value is not changed and FALSE is returned.

Location: visual.p

Parameters:

INPUT namelist AS CHARACTER

The name of one or more objects, separated by commas.

Returns: LOGICAL

Notes:

- Sets the DataModified attribute to force the toolbar to enable saving data (it behaves as if the user actually changed the field value manually).
- The DHTML client supports logical text fill-ins (HTML input) and toggle boxes (HTML check box).

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF widgetValue("status":U) = "Shipped":U THEN
    toggleWidget("transType":U).
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
if (logic.widgetValue('orderviewv.shipped')== 'Shipped')  
    logic.toggleWidget('myviewv.transtype');
```

viewWidget

Views the object or objects identified in the namelist (and their popup buttons). For SmartDataFields, it invokes the viewObject method.

Location: visual.p

Parameters:

INPUT namelist AS CHARACTER

The name of one or more objects, separated by commas.

Returns: LOGICAL

Notes: None

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF widgetIsTrue("Shipped":U) THEN  
    viewWidget("ShipDate":U).
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
if (logic.widgetIsTrue('myviewv.shipped'))  
    logic.viewWidget('orderviewv.shipdate');
```

widgetHandle

Returns the handle of the requested object. For a basic object it returns the widget-handle, for a SmartDataField it returns the procedure handle.

CAUTION: While this is not a low-level API, future versions of Dynamics may not automatically migrate calls to this API. Try to avoid using this function. If you do use it, keep track of its use so that you can quickly locate potential migration issues later.

Location: visual.p

Parameters:

INPUT name AS CHARACTER

The name of a single Dynamics object.

Returns: HANDLE

Notes: None

widgetIsBlank

Returns TRUE if the widget is blank, otherwise FALSE. If the namelist contains more than one object, then the function returns TRUE if all of them are blank, otherwise FALSE.

Location: visual.p

Parameters:

INPUT namelist AS CHARACTER

The name of one or more objects, separated by commas.

Returns: LOGICAL

Notes: None

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF widgetIsBlank("terms":U) THEN disableWidget("discount":U).
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
if (logic.widgetIsBlank('orderviewv.terms'))  
    logic.disableWidget('customerviewv.discount');
```

widgetIsFocused

Returns TRUE if the widget has focus. This is not supported for SmartDataFields. This returns unknown if the field is not found or if the widget is a SmartDataField.

Location: visual.p

Parameters:

INPUT name AS CHARACTER

The name of a single Dynamics object.

Returns: LOGICAL

Notes: This API is not supported by the DHTML client.

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF widgetIsFocused("shipped":U) AND widgetIsTrue("shipped":U) THEN  
    assignFocusedWidget("comments":U).
```

widgetIsModified

Returns TRUE if the MODIFIED attribute or its equivalent is set for the object, otherwise FALSE. If the name list contains more than one object, then the function returns TRUE if any of them have changed, otherwise FALSE. If any field in the name list is not found, unknown is returned.

For example, use this function to check if any of multiple values involved in a calculation or expression have been modified.

Location: visual.p

Parameters:

INPUT namelist AS CHARACTER

The name of one or more objects, separated by commas.

Returns: LOGICAL

Notes: None

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF widgetIsModified("deposit":U) THEN  
  assignWidgetValue("status":U, "current":U).
```

The following illustrates the use of this method in 4GL code for use with a Web browser:

```
if (logic.widgetIsModified('customerviewv.name'))  
  logic.assignWidgetValue('orderviewv.orderstatus','current');
```

widgetIsTrue

Returns TRUE if the value of a LOGICAL object is TRUE, otherwise FALSE. This function does not support SmartDataFields. This function returns unknown if the field is not found, if the value of the LOGICAL is unknown, if the widget is not a LOGICAL field, or if the widget is a SmartDataField. Contrast with the widgetValue function which returns a CHARACTER value.

Location: visual.p

Parameters:

INPUT name AS CHARACTER

The name of a single Dynamics object.

Returns: LOGICAL

Notes: None

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF widgetIsTrue("writeToConfig":U) THEN  
  RUN genConfig.
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
if (logic.widgetIsTrue('myviewv.writeToConfig'))  
    getConfig();
```

widgetValue

For most objects, returns the INPUT-VALUE of the object.

For a browse column within a ROW-DISPLAY trigger, the function returns the BUFFER-VALUE from the RowObject buffer field. If INPUT-VALUE returns a 4GL error because the value is actually blank, widgetIsBlank will be invoked and blank will be returned. For SmartDataFields, which do not have a standard function for returning an unformatted value, this function does nothing and returns unknown.

Location: visual.p

Parameters:

INPUT namelist AS CHARACTER

The name of one or more objects, separated by commas.

Returns: CHARACTER

Notes: The DHTML client supports for DATE, DECIMAL, and INTEGER data types. CHARACTER and LOGICAL data types are not supported.

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
IF widgetValue("custID":U) = cID THEN ...
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
if (logic.widgetValue('customerviewv.custnum')==cID) ...
```

widgetValueList

Takes the name of one or more objects and a delimiter and returns the INPUT-VALUE of the object. In the case of a browse column reference from within a ROW-DISPLAY event, it returns the BUFFER-VALUE from the RowObject buffer field. SmartDataFields do not have a standard function for returning an unformatted value so this function does nothing for SmartDataFields and returns unknown.

If a field in the list is not found or a field has an unknown value, the function returns “?” for its value in the returned character list.

Location: visual.p

Parameters:

INPUT `namelist` AS CHARACTER

The name of one or more objects, separated by commas.

INPUT `delimiter` AS CHARACTER

(Optional) Specifies the delimiter to use for the return value only. Specify “?” to use the default delimiter, which is the pipe character (“|”).

Returns: CHARACTER

Notes:

- The DHTML client supports DATE, DECIMAL, and INTEGER data types. CHARACTER and LOGICAL data types are not supported.
- For the DHTML client, the delimiter is optional. If it is not specified, it uses the default delimiter, which is the pipe symbol (“|”).

Examples:

The following illustrates the use of this method in 4GL code for use with a graphical user interface:

```
ASSIGN
  cValues = widgetValueList("Self.discount,balance":U, ?)
  iDiscount = INTEGER(ENTRY(1, cValues, "|":U))
  dBalance = DECIMAL(ENTRY(2, cValues, "|":U)).
```

The following illustrates the use of this method in DHTML code for use with a Web browser:

```
cValues=logic.widgetValueList('customervieww.discount,customervieww.balance');
```

Methods for data visualization objects

The following section describes the data visualization methods for visual objects.

addRecord

Procedure that 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.

Location: datavis.p

Parameters: None

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 is 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).

cancelObject

Procedure that passes a cancel request to its container.

Location: datavis.p

Parameters: None

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.

- 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.
```

cancelRecord

Procedure that 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.

Location: datavis.p

Parameters: None

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 displayed again.
 - **Update** — The Update is cancelled and the record's original values are displayed.

- 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.

canNavigate

Returns TRUE if the object can allow a navigation.

Location: datavis.p

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.

collectChanges

Procedure that collects the screen values and assigns them to the appropriate record. Values are collected as a character value of a list form.

Location: datavis.p

Parameters:

INPUT-OUTPUT pcChanges AS CHARACTER

A CHR(1)-delimited list made of changed fields and 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.

Notes:

- collectChanges is 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.)
- Use the ModifyFields property to control from which fields to collect values. See [“ModifyFields”](#) for more information.

confirmCancel

Procedure that verifies that no data is lost before allowing its container to initiate a destroy operation.

Location: datavis.p

Parameters:

INPUT-OUTPUT pLError AS LOGICAL

 If TRUE on output, there is a problem and you should cancel the destroy operation.

Notes: The name confirm is used as it is in family with the other confirm methods, but this does not ask any questions

confirmCommit

Procedure that checks to be sure there are no unsaved changes before allowing the data-source to start a commitTransaction operation.

Location: datavis.p

Parameters:

INPUT-OUTPUT pCancel AS LOGICAL

 Returns TRUE if the transaction can be committed.

Notes: None

confirmContinue

Checks to make sure there are no unsaved or uncommitted data changes before allowing its data-source to start an Apply filter operation.

Location: datavis.p

Parameters: None

Returns: LOGICAL

Notes: None

confirmDelete

Procedure that allows you to force the user to confirm a pending deletion so that it can be aborted if not intended.

Location: datavis.p

Parameters:

INPUT-OUTPUT p1Answer AS LOGICAL

Notes: None

confirmExit

Data visualization version of confirmExit. This procedure 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.

Location: datavis.p

Parameters:

INPUT p1Cancel AS LOGICAL

If field values have been modified and not saved, it returns TRUE and the destroyObject is cancelled.

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 displays 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

Procedure that verifies unsaved changes are saved and uncommitted data is committed before allowing its container to carry out a destroy operation.

Location: datavis.p

Parameters: None

Notes: The name confirm is used as it is in family with the other confirm methods, but this does not ask any questions.

confirmUndo

Procedure that checks to make sure there are no unsaved changes before allowing its data-source to start an undoTransaction operation.

Location: datavis.p

Parameters:

INPUT-OUTPUT p1Cancel AS LOGICAL

 Returns TRUE if the transaction can be undone.

Notes: None

copyRecord

Procedure that initiates creation of a new record, starting with a copy of the previously displayed record.

Location: datavis.p

Parameters: None

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.

dataAvailable

Data visualization version of dataAvailable. Procedure that requests that data columns be displayed in the Data source that the data visualization object is connected to, and displays them.

Location: datavis.p

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 displays the row in any case.

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.

deleteRecord

Procedure that 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.

Location: datavis.p

Parameters: None

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.

```

disableFields

Procedure that sets the View for ObjectMode.

Location: datavis.p

Parameters: None

Notes: Viewer and browser classes have complete override for disabling. The ObjectMode is also set in updateMode, but users might call this directly.

displayObjects

Procedure that displays values for nondatabase-related fields.

Location: datavis.i

Parameters: None

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 removes the standard version from the compilation.

displayRecord

Procedure that displays the rows for the current data source.

Location: datavis.p

Parameters: None

Notes: None

enableFields

Procedure that sets ObjectMode to Modify if the value is not Update.

Location: datavis.p

Parameters: None

Notes: Viewer and browser classes have complete override for enabling. The ObjectMode is also set in updateMode, but users might call this directly.

initializeObject

Data visualization version of initializeObject. Procedure that 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.

Location: datavis.p

Parameters: None

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.

IsUpdateActive

Procedure that is received from container source to check whether contained objects have unsaved or uncommitted changes, including addMode.

Location: datavis.p

Parameters:

INPUT-OUTPUT plActive AS LOGICAL

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.

linkStateHandler

Procedure that refreshes visual data targets.

Location: datavis.p

Parameters:

INPUT PARAMETER pcState AS CHARACTER

INPUT PARAMETER phObject AS HANDLE

INPUT PARAMETER pcLink AS CHARACTER

Notes: This version of linkStateHandler is different from the version in data.p in that queryPosition must be run in addition to dataAvailable to pass the passed QueryPosition property of the data source.

okObject

Procedure that passes an OK request to its container.

Location: datavis.p

Parameters: None

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

Procedure that checks 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.

Location: datavis.p

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.

Notes: confirmExit and confirmOpenQuery call this function. This is copied from okToContinue function in datavis.p for use with Dynamics.

queryPosition

Procedure that captures state events for the associated query in the object's data source.

Location: datavis.p

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).

Notes:

- When any state begins with 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 RecordAvailable.

resetRecord

A general code for reset operation. This procedure displays the original field values retrieved from the SmartDataObject by invoking the colValues() function for the fields that are displayed.

Location: datavis.p

Parameters: None

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 "?".

showDataMessages

Returns the name of the field (if any) from the first error message. Allows the caller to use it to position the cursor.

Location: datavis.p

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.

showDataMessagesProcedure

Procedure that 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 Progress Dynamics procedure that uses Progress Dynamics message handling routines.

Location: datavis.p

Parameters:

OUTPUT pcReturn AS CHARACTER

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.

updateMode

Procedure that enables or disables fields when the data visualization object is linked to an Update SmartPanel for which the type is **Update**.

Location: datavis.p

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.

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.

updateRecord

Procedure that receives a **Save** message from a TableIO source and sends the changed values back to the SmartDataObject. General code to start the save operation.

Location: datavis.p

Parameters: None

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.

updateState

Procedure that 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).

Location: datavis.p

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.

Notes: None

validateFields

Procedure that validates a field in a data-display object.

Location: datavis.p

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 uses this output value to set focus.

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 always checks unmodified fields. This is useful for add/copy but not otherwise, as those fields are not saved.

You can extend the base visual object to filter what it displays. The following section lists the functions and procedures for `SmartFilter` objects.

Filtering methods for visual objects

The following sections describes the methods for `SmartFilter` objects. Filtering allows users to view data in more manageable groupings.

applyFilter

Procedure that applies filter criteria to the filter-target.

Location: `filter.p`

Parameters: None

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 is 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.

blankField

Fully blanks some field and sets MODIFIED to FALSE.

Location: filter.p

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

Procedure that restores all filter fields to the empty, unmodified state.

Location: filter.p

Parameters: None

Notes: None

blankFillIn

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

Location: filter.p

Parameters:

INPUT phFillIn AS HANDLE

The handle of the fill-in to be blanked.

Returns: LOGICAL

Notes: Utility function called by blankFields and blankField.

createField

Create the Filter for one field.

Location: filter.p

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 p1Enable 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 is used to set inner-lines instead of height, and the editor has word-wrap and a vertical scrollbar.)

INPUT pdWidth AS DECIMAL

Width.

Returns: HANDLE

Notes: PRIVATE function.

createLabel

Creates the label for the filter-field.

Location: filter.p

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.

createOperator

Creates the widget to use as operator or fill-in for the range option.

Location: filter.p

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

dataAvailable

Procedure that subscribes to the filter-target in order to make sure that the filter reflects ForeignFields and values.

Location: filter.p

Parameters:

INPUT pcRelative AS CHARACTER

Not used.

Notes: The pcRelative parameter is necessary in order not to override the ForeignField criteria in the query.

dataValue

Returns the RowObject fieldname of a database fieldname.

Location: filter.p

Parameters:

INPUT pcColumn AS CHARACTER

INPUT pcValue AS CHARACTER

Returns: CHARACTER

Notes: None

deleteObjects

Deletes all dynamic widgets.

Location: filter.p

Parameters: None

Returns: LOGICAL

Notes: Called at design-time every time the Instance Properties are changed.

disableFields

Procedure that disables fields in the ENABLED-FIELDS list.

Location: filter.p

Parameters: None

Notes: None

enableFields

Procedure that enables fields in the ENABLED-FIELDS list.

Location: filter.p

Parameters: None

Notes: None

fieldLookup

Finds the LOOKUP position of the corresponding value to the field in a CHR(1) list of fields and values.

Location: filter.p

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

Finds the actual value of the corresponding value to the field in a CHR(1) list of fields and values.

Location: filter.p

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.

initializeObject

Procedure that filters object initialization code.

Location: filter.p

Parameters: None

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 assumes that CONTAINS is allowed if the filter field object type is EDITOR. UseContains also only works together with EDITOR fields. RANGE is **not** used for word-indexed fields unless UseContains is FALSE.

insertFieldProperty

Inserts or adds a fields property to the CHR(1)-separated internal property that holds fields and values.

Location: filter.p

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

removeSpace

Procedure that backspaces, deletes, and sets the field to unmodified. Keeps it out of the selection criteria.

Location: filter.p

Parameters: None

Notes: Start persistent in initializeObject. Note that filter.i has the following trigger anywhere on frame: on valueChanged > u10 > unblankFillin trigger resets format when all fields have been blanked (emptied).

resetFields

Procedure that resets all the fields to the previously applied filter values.

Location: filter.p

Parameters: None

Notes: The field values and operator values are retrieved from the actual filter-targets query.

showDataMessages

Returns the name of the field (if any) from the first error message to allow the caller to use it to position the cursor.

Location: filter.p

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.

unBlankFillin

Resets the format of a blank fill-in.

Location: filter.p

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

Procedure that resets the format of a blank fill-in of data-type LOGICAL.

Location: filter.p

Parameters:

INPUT phField AS HANDLE

INPUT phField - the handle of a field

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.

Browser methods for visual objects

The following section describes the methods for SmartDataBrowsers. SmartDataBrowsers display multiple virtual records in a simple, tabular row and column format.

addRecord

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

Location: browser.p

Parameters: None

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.

applyCellEntry

Procedure that applies **ENTRY** to the first enabled column in the browse, or to the column passed as an input parameter.

Location: browser.p

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.

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

SmartDataBrowser version of applyEntry. Procedure that applies **ENTRY** to the first enabled column in the browse if columns are enabled, or to the first enabled object in the SmartDataBrowser.

Location: browser.p

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.

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.

assignMaxGuess

Procedure that sets the argument value of the Browse widget's MAX-DATA-GUESS attribute.

Location: browser.p

Parameters:

INPUT piMaxGuess AS INTEGER

Notes: None

calcWidth

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

Location: browser.p

Parameters: None

Notes: Called by initializeObject.

cancelNew

Procedure that deletes a currently selected NEW row from the Browse's viewport.

Location: browser.p

Parameters: None

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.

cancelRecord

SmartDataBrowser version of cancelRecord. This procedure 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 displays the previously current record again. For a cancel of an update, displays the field values of the previous row again. Sets the value of the NewRecord property back to No.

Location: browser.p

Parameters: None

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.

colValues

Formats the values in the current row of the SmartDataBrowser query for the specified column list. SmartDataBrowser version of colValues.

Location: browser.p

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).

copyRecord

Procedure that initiates creation of a new record starting with a copy of the previously displayed record. SmartDataBrowser version of copyRecord.

Location: browser.p

Parameters: None

Notes:

- Invokes copyRecord in datavis.p (RUN SUPER), which in turn invokes the copyRow function in the associated SmartDataObject to add a new row to the RowObject temp-table.
- For a SmartDataBrowser, a new blank row is inserted below the currently selected one. Values that are copied from copyRecord are not displayed here, they are displayed from the ROW-ENTRY trigger (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.

dataAvailable

SmartDataBrowser version of dataAvailable. This procedure repositions the SmartDataBrowser to a specific row in response to a query reposition in its data source.

Location: browser.p

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.

Notes:

- You can override the version of dataAvailable in browser.p.
- 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.

defaultAction

Procedure that runs persistently from default action.

Location: browser.p

Parameters: None

Notes: The trigger is defined in setActionEvent.

deleteComplete

Procedure that deletes the currently selected row from the viewport of the browse when that row has been deleted from the RowObject result set.

Location: browser.p

Parameters: None

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 is invoked from deleteRecord, which in this case does nothing as the row was already deleted.

deleteRecord

Procedure that deletes the current record. SmartDataBrowser version of deleteRecord. Invoked when the delete button is chosen in an Update SmartPanel or in a SmartToolbar.

Location: browser.p

Parameters: None

Notes: Invokes deleteRecord in datavis.p (RUN SUPER), which then runs deleteRow in the associated SmartDataObject to delete the row from the RowObject result set. 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.

destroyObject

SmartDataBrowser version of destroyObject. This procedure invokes the standard behavior in datavis.p and as a result, destroyObject informs the SmartDataObject (and any attached SmartDataBrowser) that it is no longer being browsed.

Location: browser.p

Parameters: None

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.

disableFields

Procedure that sets the READ-ONLY attribute of the browse control to TRUE. This is the SmartDataBrowser version of disableFields.

Location: browser.p

Parameters:

INPUT PcFields AS CHARACTER

Acceptable values: **All** or **Create**.

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.

displayFields

Procedure that 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.

Location: browser.p

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.

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 displaying the current row again.
- 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.

enableFields

SmartDataBrowser version of enableFields. This procedure enables columns in a SmartDataBrowser by turning off the READ-ONLY attribute for the browse object.

Location: browser.p

Parameters: None

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.

fetchDataSet

Procedure that changes or resets the state of a Browse, depending on which action occurs.

Location: `browser.p`

Parameters:

INPUT pcState AS CHARACTER

The character value of one of the supported states. Possible values include: **BatchStart**, **BatchEnd**, **NextStart**, **NextEnd**, **PrevStart**, **PrevEnd**.

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.

filterActive

Procedure that calls updateState in the toolbar to set the filter on or off, depending on the argument.

Location: `browser.p`

Parameters:

INPUT plActive AS LOGICAL

TRUE if the filter tick is to be turned on.

Notes: None

hasActiveAudit

If there is a valid DataSource, return the result of hasActiveAudit.

Location: `browser.p`

Parameters: None

Returns: LOGICAL

Notes: None

hasActiveComments

If there is a valid DataSource, return the result of hasActiveComments.

Location: browser.p

Parameters: None

Returns: LOGICAL

Notes: None

initializeObject

SmartDataBrowser version of initializeObject. Part of the general initialization process, this procedure sets certain property values.

Location: browser.p

Parameters: None

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.

launchFolderWindow

Procedure that calls launchContainer in the session manager to start a folder window, if there is not already one available.

Location: browser.p

Parameters:

INPUT pcContainerMode AS CHARACTER

Notes: None

offEnd

Procedure that fires trigger code for OFF-END trigger of SmartDataBrowse.

Location: browser.p

Parameters: None

Notes: None

offHome

Procedure that fires trigger code for OFF-HOME trigger of SmartDataBrowse.

Location: browser.p

Parameters: None

Notes: None

onEnd

Event procedure for the END event. This procedure uses the QueryPosition property of the dataSource to determine whether the position is already last or whether fetchLast needs to be called in the dataSource.

Location: browser.p

Parameters: None

Notes: This event must call fetchLast in the dataSource for the normal case when it is a TRUE user event. However, APPLY END must be used to make the browse respond to a fetchLast in the dataSource. When using APPLY END, QueryPosition is already set in the dataSource and can be used to avoid an extra or endless loops.

onHome

Event procedure for the HOME event. This procedure uses the QueryPosition property of the dataSource to determine whether the position is already first or whether fetchFirst needs to be called in the dataSource.

Location: browser.p

Parameters: None

refreshBrowse

Procedure that publishes from dataSource when APPEND display is not enough or not required.

Location: browser.p

Parameters: None

Notes: None

resetRecord

SmartDataBrowser version of resetRecord. This procedure redisplay the original field values retrieved from the SmartDataObject.

Location: browser.p

Parameters: None

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.

resizeBrowse

Procedure that changes the size of the BROWSER WIDGET-HANDLE IN APPEND SmartDataBrowser.

Location: browser.p

Parameters:

INPUT pd_height AS DECIMAL

The desired height (in rows).

INPUT pd_width AS DECIMAL

The desired width (in columns).

Notes: None

resizeObject

Procedure that changes the size of the SmartDataBrowser.

Location: browser.p

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.

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.

rowChanged

Event handler for value-Changed procedure of browse anywhere event.

Location: browser.p

Parameters: None

Notes: This procedure is not reliable fired on record changes. You should use dataAvailable if you want to perform this type of event.

rowDisplay

Event Handle procedure for the ROW-DISPLAY event.

Location: browser.p

Parameters: None

Notes: None

rowVisible

Determines if the records currently visible in the browse are the first or last records in the batch.

Location: browser.p

Parameters:

INPUT pcRowids AS CHARACTER

Comma-separated list of ROWIDs that are currently visible.

INPUT phQryBuffer AS HANDLE

Returns: CHARACTER

Notes: None

searchTrigger

Procedure that 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.

Location: browser.p

Parameters: None

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.

startSearch

Procedure that launches the start-search trigger.

Location: browser.p

Parameters:

INPUT phBrowse AS HANDLE

The browse handle.

Notes: Because persistent triggers have early binding, current-column cannot be passed in.

stripCalcs

Strips the expression portion of expression @ variable.

Location: browser.p

Parameters:

INPUT cClause AS CHARACTER

The clause to be edited

Returns: CHARACTER

Notes: None

toolbar

Procedure that performs one of several operations depending on the value of the argument passed. 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**.

Location: browser.p

Parameters:

INPUT pcValue AS CHARACTER

The type of operation

Notes: None

updateRecord

SmartDataBrowser version of updateRecord. This procedure 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.

Location: browser.p

Parameters: None

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.

updateState

Procedure that 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.

Location: browser.p

Parameters:

INPUT PcState AS CHARACTER

The update state value to be passed to the SmartDataBrowser. Valid values are **UpdateBegin**, **Update**, and **UpdateComplete**.

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.

updateTitle

Procedure for the SmartDataBrowser version of updateTitle.

Location: browser.p

Parameters: None

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.

ValueChanged

Event procedure that captures changes to values in columns and sets the DataModified flag to indicate that data must be saved.

Location: browser.p

Parameters: None

Notes: This is not the same as the event handler of the browse. The browse event handler is **onValueChanged**.

viewObject

Procedure that 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.

Location: browser.p

Parameters: None

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.

Methods for SmartDataViewers

The following section describes the methods for SmartDataViewers. SmartDataViewers display a single virtual record at a time using a combination of fill-in fields and SmartDataFields.

addRecord

Procedure that 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.

Location: viewer.p

Parameters: None

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.

cancelRecord

Procedure that cancels an Update, Add, or Copy operation.

Location: viewer.p

Parameters: None

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.

copyRecord

Procedure that initiates creation of a new record starting with a copy of the previously displayed record.

Location: viewer.p

Parameters: None

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.

disableFields

Procedure that disables the fields in a list represented by the AppBuilder generated preprocessor value {&ENABLED-FIELDS}.

Location: viewer.p

Parameters:

INPUT *pcFieldType* AS CHARACTER

Can have values of **All** or **Create**.

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.

displayFields

Procedure that displays the current row values by moving them to the viewer's frame screen-values.

Location: viewer.p

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.

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.
- No RowObject record is available in a SmartDataViewer, so a customization of displayFields cannot refer to the RowObject buffer. You can use the colValues function to access a list of column values from the Viewer's SmartDataObject if needed, or the columnStringValue function can 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.

```

enableFields

Procedure that enables fields in the ENABLED-FIELDS list AppBuilder generated preprocessor.

Location: viewer.p

Parameters: None

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.

fieldModified

Event procedure that handles the setting of DataModified when a field has changed in the datavisual (viewer or browser) object. The setting for DataModified is determined by the setting specified for the ModifyFields property.

Location: viewer.p

Parameters:

INPUT phField AS HANDLE

Handle of a field whose value has changed.

Notes:

- By default, DataModified is set to TRUE. You can change the default using the "ModifyFields" property.
- The passed field must be in the list of EnabledFields or EnabledObjFlds.

- This procedure is called from valueChanged, DataModified of the SmartDataField, or the Client APIs that change widget values.
- Returns TRUE if this change triggers a DataModified change. For example, this is the first change in the object.

initializeObject

Procedure that enables fields in the list represented by the AppBuilder generated preprocessor value {&ENABLED-FIELDS}.

Location: viewer.p

Parameters: None

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.

linkState

Receives messages and republishes events when a GroupAssignTarget becomes **active** (when it is viewed) or **inactive** (when it hidden).

Location: viewer.p

Parameters:

INPUT pcState AS CHARACTER

Active or Inactive — The event is republished up the groupAssignSource and the DataSource which then runs linkStateHandler if the link cannot be deactivated.

Notes: None

linkStateHandler

Runs this version of linkStateHandler procedure with the same parameter for all GroupAssignTargets. As a result, all group assigns are synchronized.

Location: viewer.p

Parameters:

INPUT pcState AS CHARACTER

INPUT pcLink AS CHARACTER

Linkname for the data source.

INPUT pcObject AS HANDLE

Object to which you want to link.

Notes: None

toolbar

Toolbar event handler procedure that handles the cases **OK** and **Cancel**.

Location: viewer.p

Parameters:

INPUT pcValue AS CHARACTER

Valid values: **OK**, **Cancel**

Notes: None

updateState

Procedure that initializes code specific to SmartDataViewers.

Location: viewer.p

Parameters: None

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.

valueChanged

Procedure that defines the persistent event handler of the value-changed event of the frame. This handler is important to how a viewer and browser sets DataModified to TRUE. Because all field-level objects value-changed events are also managed by this procedure, it calls fieldModified to set DataModified to TRUE based on the setting for the ModifyFields property.

Location: viewer.p

Parameters: None

Notes:

- Static objects can reach this also by applying U10 event to run it.

Notes: See [“ModifyFields”](#) for more information.

viewRecord

Procedure that sets object in viewMode.

Location: viewer.p

Parameters: None

Notes: None

Methods for TreeView objects

The following section describes the methods for TreeView objects.

addNode

Procedure that adds a node to the tree.

Location: treeview.p

Parameters:

INPUT phBuffer AS HANDLE

Handle to buffer of temp-table tTreeData used to defined the node.

The following table lists and describes the fields and temp-table includes defined in TVController.

Table 3–1: Temp-table includes for fields defined in TVController (1 of 2)

Field	Temp-table includes
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.

Table 3–1: Temp-table includes for fields defined in TVController*(2 of 2)*

Field	Temp-table includes
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.

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

deleteNode

Procedure that removes an individual node from the tree.

Location: treeview.p

Parameters:

INPUT pckey AS CHARACTER

The unique key that identifies the node.

Notes: None

deleteTree

Procedure that can be called to remove nodes from the Tree.

Location: treeview.p

Parameters:

INPUT phTable AS HANDLE

The handle of tTreeData temp-table (defined in TVController.i).

Notes: This differs from the deleteNode procedure in that the tempTable can specify multiple nodes to delete at one time.

disableObject

TreeView version of disableObject procedure.

Location: treeview.p

Parameters: None

Notes: None

emptyTree

Procedure that clears all nodes on the TreeView.

Location: treeview.p

Parameters: None

Notes: None

enableObject

TreeView version of enableObject procedure.

Location: treeview.p

Parameters: None

Notes: None

isNodeExpanded

Returns TRUE if some node is expanded.

Location: treeview.p

Parameters:

INPUT pcNodeKey AS CHARACTER

The node in question

Returns: LOGICAL

Notes: None

loadImage

Returns the index of some image file in the ImageList, first loading it if necessary. If the load fails, the function returns 0 (zero).

Location: treeview.p

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 image list does not contain any images, the ImageList property is assigned to the TreeView only once, after an image is added.

populateTree

Procedure that controls the population of the TreeView. Creates the node given in the argument and all its descendents.

Location: treeview.p

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.

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).

repositionObject

Procedure used for super override for TreeView object repositioning.

Location: treeview.p

Parameters:

INPUT pdRow AS DECIMAL

New row.

INPUT pdCol AS DECIMAL

New column.

Notes: None

resizeObject

Procedure that resizes a SmartTreeView object.

Location: treeview.p

Parameters:

INPUT pdHeight AS DECIMAL

New height.

INPUT pdWidth AS DECIMAL

New width.

Notes: None

selectFirstNode

Procedure that selects the first node in the TreeView.

Location: treeview.p

Parameters: None

Notes: None

selectNode

Switches focus to node passed in by phNodeKey, selects this node, and generates a tvNodeSelected event.

Location: treeview.p

Parameters:

INPUT pcNodeKey AS CHARACTER

Node to receive focus

Returns: LOGICAL

Notes: None

showTVError

Procedure that displays the error message passed as the argument.

Location: treeview.p

Parameters:

INPUT pcMessage AS CHARACTER

The error message to display

Notes: None

updateTree

Procedure that updates details of nodes that are currently in the tree.

Location: treeview.p

Parameters:

INPUT phTable AS HANDLE

Handle of tTreeData

Notes: None

Visual object properties

Visual Object properties provide information about visual objects and their classes. This information can include whether an object is enabled, the contents of the object and so on. You can read property values and in many instances you can change property values. To read a value for a property, you use a **get** function, and to change a value for a property, you use a **set** function.

These functions conform to the following conventions:

- **get** — Uses the form *getpropnam* and returns the current value of the property

NOTE: This function accepts no arguments.

- **set** — Uses the form *setpropname*. The set function accepts a single argument—the new value for the property—and returns TRUE/FALSE depending on whether the value change succeeds.

For more information about getting and setting property values, see [Chapter 1, “ADM2 SmartObject API Reference”](#) in this guide.

AllFieldHandles

This property is defined for all visual objects and returns a list of handles for all widgets in the SmartObject's frame.

Data Type: CHARACTER

Notes:

- Read and Write
- This property is used only for security in the Datavis class. It should not be used in the Field class.

AllFieldNames

This property is defined for all visual objects and returns a list of all widgets in the SmartObject's frame. This list of widgets includes data fields that come back as part of the DisplayFields list and other objects such as buttons and fields that are not derived from an SDO, including rectangles and fields labels that are separate widgets with their own handles.

Data Type: CHARACTER

Notes:

- Read and Write
- For SmartDataFields, the logical name of the dynamic Lookup or Combo is returned in the AllFieldNames list along with its procedure handle in the AllFieldHandles list.
- This property is used only for security in the Datavis class. It should not be used in the Field class.

ApplyActionOnExit

Currently used by SmartSelect, this property indicates whether or not exit is to perform the same action as the DEFAULT-ACTION. Set to TRUE if exit is to perform the same action. Currently used by SmartSelect.

Data Type: LOGICAL

Notes: Read and Write

ApplyExitOnAction

Currently used by SmartSelect, this property indicates whether or not the DEFAULT-ACTION is to exit the browse. Set to TRUE if DEFAULT-ACTION is to exit the browse.

Data Type: LOGICAL

Notes: Read and Write

AutoSort

Indicates whether the root node should be auto-sorted.

Data Type: LOGICAL

Notes: Read and Write

BrowseHandle

Handle of the browse control.

Data Type: HANDLE

Notes: Read only

CalcWidth

Logical value that determines whether the width of the browse is calculated to the exact amount needed for the fields it contains.

Data Type: LOGICAL

Notes: Read and Write

ClientRect

Returns the client rectangle for the TreeView.

Data Type:

Notes: Read only

Col

Column of the object.

Data Type: DECIMAL

Notes: Read only

color3DFace

Property that can be used to set BGCOLOR or FGCOLOR on any Progress three-dimensional face widget.

Data Type: Integer

Notes: Read only

color3DHighLight

Property that can be used to set BGCOLOR or FGCOLOR on any Progress three-dimensional highlight widget.

Data Type: Integer

Notes: Read only

color3DShadow

Property that can be used to set BGCOLOR or FGCOLOR on any Progress three-dimensional shadow widget.

Notes: Read only

CreateHandles

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.

Data Type: CHARACTER

Notes: Read only

CtrlFrameHandle

Handle of the TreeView's control frame.

Data Type: HANDLE

Notes: Read only

DataModified

Indicates whether the current SCREEN-VALUES have been modified but not saved.

Data Type: LOGICAL

Notes:

- Read and Write
- When DataModified is set to TRUE, the check occurs before any other code is executed.
- Disables searching while there is an update in progress in the SmartDataBrowser. SmartDataBrowser version of setDataModified.

DataObject

Filter used at design time for data objects.

Data Type: CHARACTER

Notes: Read and Write

DataSignature

A character string that lists the integers corresponding to the datatypes of the fields in an object temp-table. This string is used to compare objects for equivalence as follows:

<pre>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</pre>
--

DefaultCharWidth

Default width of character fields.

Data Type:

Notes: Read and Write

DefaultEditorLines

Default number of inner lines for editors.

Data Type: DECIMAL

Notes: Read and Write

DefaultLayout

The default layout for the object.

Data Type: CHARACTER

Notes: Read only

DefaultWidth

Default width for non-character fields.

Data Type: DECIMAL

Notes: Read and Write

DisableOnInit

Indicates whether the current object should be left disabled when it is first initialized.

Data Type: LOGICAL

Notes: Read and Write

DisplayedField

Name of the field to display in the selection. This property can be used in signature matching.

Data Type: CHARACTER

Notes: Read and Write

DisplayFieldsSecurity

Comma-separated list of security types corresponding to AllFieldHandles.

Data Type: CHARACTER

Notes: Read and Write

DisplayedTables

List of SDO table names used by the visualization. Can be only **RowObject**. If the object was built against an SBO, it is the list of SDO ObjectNames whose fields are used.

Data Type: CHARACTER

Notes: Read only

Down

Down attribute for the browse. Called from initializeObject for dynamic SmartDataBrowsers.

Data Type: INTEGER

Notes: Write only

Editable

Indicates whether this object can be edited (add/copy/save/update).

Data Type: LOGICAL

Notes: Read and Write

EnabledObjFlds

List of the field names of widgets enabled in this object not associated with data fields.

Data Type: CHARACTER

Notes: Read only

EnabledObjFldsToDisable

Property that controls whether or not to disable non-database objects when the data fields are disabled. You can enter one of the following:

- **None** — Non-database objects remain enabled when the fields are disabled.
- **All** — Non-database objects are disabled in view mode
- **Comma separated list** — A comma-separated list of non-database object names that you want disabled in view mode.

Data Type: CHARACTER

Notes:

- Read and Write
- The property only applies to non-database objects that have been defined as enabled in the master. See [EnabledObjFlds](#) for more information about enabling.
- You can edit this property using the viewer's Instance Property dialog box.

EnabledObjHdls

List of the handles of widgets enabled in this object and not associated with data fields.

Data Type: CHARACTER

Notes: Read only

EnabledFields

A comma-separated list of the names of enabled fields in this object that map to fields in the SmartDataObject (&ENABLED-FIELDS).

Data Type: CHARACTER

Notes:

- Read and Write.
- When EnabledFields is set to FALSE, the check occurs before other code is executed.

EnabledHandles

A comma-separated list of the handles, in character format, of the enabled fields in the visualization object.

Data Type: CHARACTER

Notes: Read and Write

ExpandOnAdd

Use this property along with the TreeView property to automatically expand a node that has been newly added to a TreeView.

Data Type: LOGICAL

Notes: Read and Write

FieldColumn

Column number of the left most field.

Data Type: DECIMAL

Notes: Read and Write

FieldFormats

Internal override of formats for fields.

Data Type: CHARACTER

Notes: Read and Write

FieldHandles

A comma-separated list of the handles, in character format, of the Data Fields in the visualization object.

Data Type: CHARACTER

Notes: Read and Write

FieldHelpIds

Internal list of properties for the fields. Each element consists of a name and a value, separated by CHR(1).

Data Type: CHARACTER

Notes: Read and Write

FieldsEnabled

Indicates whether the fields in this visualization object are enabled.

Data Type: LOGICAL

Notes: Read only

FieldLabels

Internal list of properties for the fields. Each element consists of a name and a value, separated by CHR(1).

Data Type: CHARACTER

Notes: Read and Write

FieldOperatorStyles

Internal list of properties for the fields. Each element consists of a name and a value, separated by aCHR(1)

Data Type: CHARACTER

Notes: Read and Write

FieldToolTips

Internal list of properties for the fields. Each element consists of a name and a value, separated by a CHR(1).

Data Type: CHARACTER

Notes: Read and Write

FieldWidths

Internal list of properties for the fields. Each element consists of a name and a value, separated by a CHR(1).

Data Type: CHARACTER

Notes: Read and Write

FilterActive

Values of FilterActive and QueryColumns. TRUE if a filter is active and QueryColumns is not the empty string and TRUE if the DataSource has a logical filter.

Data Type: LOGICAL

Notes:

- Read only
- It might be set to TRUE explicitly or use the Querycolumns

FilterTarget

Handle for a data object, in character format or the name of the linked filter object. Currently supports only one.

Data Type: CHARACTER

Notes: Read and Write

FilterTargetEvents

Comma-separated list of the events to which this object wants to subscribe to in its FilterTarget.

Data Type: CHARACTER

Notes: Read and Write

FolderWindowToLaunch

Property that contains the name of a logical object (usually a folder window) to launch when a user selects a maintenance option on an object controller (usually a container and a browser with a toolbar).

Use this property to launch an object that allows the user to maintain details for records.

Data Type: CHARACTER

Notes: Read only

FrameMinHeightChars

Predetermined character height of a frame.

Data Type: DECIMAL

Notes:

- Read and Write
- This value is not saved when a dynamic viewer is resized in design mode. The value is calculated based on the position of widgets on the viewer,
- Translations applied at runtime cause this value to be recalculated.

FrameMinWidthChars

Predetermined character width of a frame.

Data Type: DECIMAL

Notes:

- Read and Write
- This value is not saved when a dynamic viewer is resized in design mode. The value is calculated based on the position of widgets on the viewer,
- Translations applied at runtime cause this value to be recalculated.

FullRowSelect

The FullRowSelect property of the tree. If TRUE, the entire node (text and icon) is highlighted.

Data Type: LOGICAL

Notes: Read and Write

GroupAssignHidden

Property used to verify whether the object is hidden and if it can remain hidden.

Data Type: Logical

Notes:

- Read only
- This property is used by the linkState procedure in the `data.p` file to verify that the link can be deactivated and how to republish the message.
- Returns FALSE as soon as any visible object is found.
- This property is also used to verify whether all GroupAssignTargets are hidden and if they should remain hidden.

GroupAssignSource

Handle of the object's GroupAssign source.

Data Type: HANDLE

Notes: Read and Write

GroupAssignSourceEvents

Comma-separated list of the events this object wants to subscribe to in its GroupAssign source.

Data Type: CHARACTER

Notes: Read and Write

GroupAssignTarget

Handle, in character format, of the object's GroupAssign target.

Data Type: CHARACTER

Notes: Read and Write

GroupAssignTargetEvents

Comma-separated list of the events this object wants to subscribe to in its GroupAssign target.

Data Type: CHARACTER

Notes: Read and Write

Height

Height of the object.

Data Type: DECIMAL

Notes: Read only

HideOnInit

Indicates whether the current object should be left hidden when it is first initialized.

Data Type: LOGICAL

Notes: Read and Write

HideSelection

Indicates whether the Tree View remains highlighted. If TRUE, the current node in the TreeView does not remain highlighted when focus leaves the Tree View.

Data Type: LOGICAL

Notes: Read and Write

ImageHeight

Height of images in the image list associated with the TreeView.

Data Type: INTEGER

Notes: Read and Write

ImageWidth

Width of images in the image list associated with the TreeView.

Data Type: INTEGER

Notes: Read and Write

ILComHandle

The com-handle of the ImageList ActiveX object.

Data Type: HANDLE

Notes: Read only

Indentation

Number of pixels of indentation between two generations of nodes.

Data Type: INTEGER

Notes: Read and Write

KeepChildPositions

Determines if widget positions should change to accommodate changes in their labels. If TRUE, the translated label is truncated, if necessary, to fit the space occupied by the original label. The full translated label is available in the label's tooltip. If FALSE, the widget is repositioned to allow enough space for a translated label.

Data Type: LOGICAL

Notes: Read and Write

LabelEdit

Indicates whether a user can edit a node. If TRUE, the user can click on the node to edit it.

Data Type: INTEGER

Notes: Read and Write

LayoutOptions

List of multi-layout options for the object.

Data Type: CHARACTER

Notes: Read and Write

LayoutVariable

Name of the &LAYOUT-VARIABLE preprocessor for the object, which is used as a prefix to the name of the procedure that resets it.

Data Type: CHARACTER

Notes: Read only

LineStyle

Determines the line style for a tree. This property has the following options:

- 0 (Default) — Displays lines between child nodes and their parent.
- 1 — Displays lines between root nodes and lines between child and parent nodes.

Data Type: INTEGER

Notes: Read and Write

MaxWidth

Maximum width of the browse when CalcWidth is TRUE.

Data Type: DECIMAL

Notes: Read and Write

MinHeight

Determines the minimum height.

Data Type: DECIMAL

Notes: Read and Write

MinWidth

Determines the minimum width.

Data Type: DECIMAL

Notes: Read and Write

ModifyFields

Property that determines which fields should set DataModified to TRUE in their containing viewer when fields are modified. For data-source fields, this property also controls the fields from which values are collected for passing to the data source and update target.

Valid values for this property are:

- **All** — All EnabledObjects should set DataModified.
- **None** — No EnabledObjects should set DataModified.
- **EnabledFields** — Only Fields from the data source should set DataModified.
- **Updatable** — Only EnabledFields that can be updated in the data source should set DataModified.
- **Comma separated list** — List of object names that can set DataModified when changed.

Data Type: CHARACTER

Notes:

- Read and Write.
- Setting DataModified to TRUE in the viewer enables Save and Reset in the TableioSource toolbar.
- DataFields that do not set DataModified using one of these values are not passed to the update target on save. However, the data source value is displayed again on save.

- ModifyFields is used to determine which field values are collected by the “collectChanges” procedure.
- You can edit this property using the viewer's Instance Property dialog box.

NextNodeKey

Unique code to be used for the Key attribute of the Tree view.

Data Type: CHARACTER

Notes:

- Read only
- You cannot begin a key for a node with a number in Microsoft TreeView. Therefore, the prefix xcNodePrefix is used.
- The key is used as the sort field when adding nodes. Therefore, the string “999999999999” is used to guarantee a valid sort order.

NodeExpanded

Expands or collapse a node based on the argument passed.

Data Type: LOGICAL

Notes: Write only

NumDown

Number of rows that are displayed DOWN in the browse.

Data Type: INTEGER

Notes: Read and Write

ObjectEnabled

Indicate whether the current object is enabled.

Data Type: LOGICAL

Notes: Read only

ObjectLayout

Current layout name for the object.

Data Type: CHARACTER

Notes: Read and Write

OLEDrag

Determines whether OLEDrag is supported. If TRUE, drag is supported.

Data Type: LOGICAL

Notes: Read and Write

OLEDrop

Determines whether OLEDrop is supported. If TRUE, drop is supported.

Data Type: LOGICAL

Notes: Read and Write

Operator

The Operator, the default when OperatorStyle is **Implicit**.

Data Type: CHARACTER

Notes: Read and Write

OperatorLongValues

List of operators and long text.

Data Type: CHARACTER

Notes: Read and Write

OperatorStyle

The style for the operator. The following lists the valid values for operator:

- **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).

Data Type: CHARACTER

Notes: Read and Write

OperatorViewAs

View-as type used to define the operator when OperatorStyle equals **Explicit**.

Data Type: CHARACTER

Notes: Read and Write

Property

Properties for the specified 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

Data Type: CHARACTER

Notes: Read and Write

QueryRowObject

Handle of the RowObject temp-table associated with the Browse's query.

Data Type: HANDLE

Notes: Read and Write

RecordState

String indicating whether a record is available or not. Valid values are **RecordAvailable** and **NoRecordAvailable**.

Data Type: CHARACTER

Notes: Read only

Refresh

Refreshes the TreeView.

Data Type:

Notes: Write only

ResizeHorizontal

Indicates whether an object can be resized horizontally. The value for this property is TRUE if the object can be resized horizontally (in X).

Data Type: LOGICAL

Notes: Read and Write

ResizeVertical

Indicates whether an object can be resized vertically. The value for this property is TRUE if the object can be resized vertically (in Y).

Data Type: LOGICAL

Notes: Read and Write

RootNodeParentKey

Parent key for root nodes on the tree. You can use the parent key to find all root nodes that has this parent.

Data Type: CHARACTER

Notes: Read only

Row

Height of the object. Use repositionObject to set the Row.

Data Type: DECIMAL

Notes: Read only

RowIdent

The ROWID, in character format, of the current row of the visualization. If the update-target is an SmartDataObject (SDO), the database ROWIDs can be stored as the second entry of the list. If connected to a SmartBusinessObject (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.

Data Type: CHARACTER

Notes:

- Read and Write
- Returns 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.

RowObject

Handle of the RowObject Temp-Table buffer or the Browser's temp-table definition.

Data Type: HANDLE

Notes: Read only

SaveSource

Indicates whether the TableIO-Source is set to Save or Update. This value is TRUE if the TableIO-Source is set to Save and FALSE if set to Update (modal).

Data Type: LOGICAL

Notes: Read and Write

Scroll

Scroll property of the tree. If TRUE, scrollbars appear.

Data Type: LOGICAL

Notes: Read and Write

ScrollRemote

Value of ScrollRemote.

Data Type: LOGICAL

Notes: Read and Write

SearchField

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.

Data Type: CHARACTER

Notes: Read and Write

SelectedNode

Key of the selected node.

Data Type: CHARACTER

Notes: Read only

ShowCheckBoxes

Indicates whether a check box appears next to a node. If TRUE, check boxes appear beside each node on the TreeView.

Data Type: LOGICAL

Notes: Read and Write

ShowRootLines

Indicates whether there should be lines to the root of the tree. If TRUE, there should be lines leading to the roots of the tree.

Data Type: LOGICAL

Notes: Read and Write

SingleSel

Controls whether a node is expanded when selected. If TRUE, the node is expanded when selected or clicked.

Data Type: LOGICAL

Notes: Read and Write

Sort

Identifies whether a selection should be sorted. If TRUE, the selection is to be sorted.

Data Type: LOGICAL

Notes: Read only

TableIOSource

Identifies the source for the input and output Table.

Data Type: HANDLE

Notes: Read and Write

TableIOSourceEvents

Comma-separated list of the events to which this object wants to subscribe to in its TableIO source.

Data Type: CHARACTER

Notes: Read and Write

ToggleDataTargets

Property used to control whether or not dataTargets should be set to **on** or **off** in linkState. This value changes based on what is passed with the active or inactive parameter.

Data Type: Logical

Notes:

- Read and Write
- If TRUE, dataTargets should be set to on or off in linkState depending on what is passed with the active or inactive parameter.
- If FALSE, dataTargets should **not** be set to on or off in linkState depending on what is passed with the active or inactive parameter.

ToolbarSource

Handle or list of handles.

Data Type: CHARACTER

Notes: Read and Write

ToolbarSourceEvents

List of events to be subscribed to in the Toolbar-Source.

Data Type: CHARACTER

Notes: Read and Write

ToolTip

Property containing the ToolTip for the browser.

Data Type: CHARACTER

Notes:

- Read and Write
- This property is supported in both Dynamics and Non-Dynamics environments.

TreeDataTable

Handle of a dynamic temp-table populated with property fields for a TreeView.

Data Type: HANDLE

Notes: Read only

TreeStyle

Use this property with the TreeView property to specify the appearance for a TreeView. You can specify the following to define the appearance of a Tree View:

- **0** — Text only.
- **1** — Image and text.
- **2** — Plus and minus with text.
- **3** — Plus and minus with image and text.
- **4** — Lines and text.
- **5** — Lines, image, and text.
- **6** — Lines, plus and minus, with text.
- **7** — (Default) Lines, plus and minus with image and text.

Data Type: INTEGER

Notes: Read and Write

TVControllerSource

Handle of the procedure controlling this TreeView.

Data Type: HANDLE

Notes: Read and Write

TVControllerTarget

Usually, the handle of the TreeView being controlled.

Data Type: HANDLE

Notes: Read and Write

TVControllerTargetEvents

List of target events to which this controller wants to subscribe.

Data Type: CHARACTER

Notes: Read only

UpdateTarget

Used for pass-through links, to connect an object inside the container with an object outside the container.

Data Type: CHARACTER

Notes: Read and Write

UpdateTargetNames

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.

Data Type: CHARACTER

Notes: Read and Write

UseBegins

Value of UseBegins. This value is TRUE when BEGINS is supposed to be used for character tests.

Data Type: LOGICAL

Notes: Read and Write

UseContains

Property used to determine when to use Contains. If TRUE, then Contains should be used as operator for character values.

Data Type: LOGICAL

Notes: None

ValidKey

Indicates whether there is a valid key. If TRUE, there is a valid key.

Data Type: LOGICAL

Notes: Read only

VisualBlank

Used to visualize searching for BLANK values.

Data Type: CHARACTER

Notes: Read and Write

Width

Width of the object.

Data Type: DECIMAL

Notes: Read only

WindowTitleField

This property usually contains the name of a field that you want to concatenate to the window title of a container. You can use this property to indicate what record is being maintained.

For example, if you have an object controller for Products, and you launch a folder window and you want the product code to appear in the title, you would assign the value **gsc_product.product_code** to this property. As a result, when the folder window launches, the product code of the product being maintained is concatenated to the window title.

Data Type: CHARACTER

Notes: Read and Write

Column properties for visual objects

There are a number of column properties available for which you can obtain and set (write) field values. All of these properties can be read and some of them can be set. To read and set column properties, you use the following prefixes with the specific column property:

- **Column** — Use to read the value of a specific column property. For example, if you want to read the value of the FilterTarget property, you would specify **FilterTarget**. This would return the label of the column you specify. To obtain the data type for a specific column property, you would specify **DataType**. This would return the data type of the specified column.
- **Assign** — Use to set the value of the specified column property.

NOTE: For Container objects, there are no column properties for which you can assign the value.

For additional information, see the *Progress Dynamics Programming Handbook*.

[Table 3–2](#) lists and provides a brief description of the column properties for field objects:

Table 3–2: Column properties for visual objects

Column property	Description	Read	Write	Data type
DataType	Data type of the specified column.	Yes	No	CHARACTER
FilterTarget	Filter-Target of the specified column.	Yes	No	HANDLE
Format	Format text of a specified column.	Yes	Yes	CHARACTER
HelpID	HelpID for a column.	Yes	Yes	INTEGER
Label	Label text for a column	Yes	Yes	CHARACTER
OperatorStyle	Operator style for some field, derived either from FieldOperatorStyles or the OperatorStyle property	Yes	Yes	CHARACTER
StyleDefault	Name of the column in the filter-target. Used only in design-time because the columnOperatorStyle function returns the overridden value. TRUE if the style has been overridden.	Yes	No	LOGICAL
Tooltip	Tooltip for a column.	Yes	Yes	CHARACTER
Width	Width (in character units) of a specified column.	Yes	Yes	DECIMAL
WidthDefault	Indicates whether or not a column width has been overridden. TRUE if the width has been overridden.	Yes	No	DECIMAL

Container Objects and Their Methods and Properties

This chapter lists and describes the methods (internal procedures and functions) and properties used for Container Objects. Refer to [Figure 1–1](#) to see the inheritance hierarchy for each object class.

NOTE: For information specific to the WebSpeed environment (see the [“Alphabetical Listing Of WebSpeed-specific API Routines”](#) chapter).

Base methods for container objects

The following section describes the base methods for Container Objects.

addDataObject

Constructs and links a data object, and returns the handle of the started object.

Location: `dynacontainer.w`

Parameters:

`pcPhysicalObject` AS CHAR

Physical relative path name of the DataObject.

`pcName` AS CHAR

Name of the Object instance. If the `physicalName` is specified, then this parameter gives it an instance name for the container. It can also be used to specify that this call is to add new properties or a link for an object that was added previously with this function. If no physical name is given, and the name does not exist in the container, then this is used as the `LogicalObjectName` to fetch the information from the Repository. **Unknown** can be specified to tell the container to assign an appropriate instance name.

`phParent` AS HANDLE

Handle of an object to link as a data source to this object.

`pcForeignFields` AS CHAR

Foreign fields to use for the specified data source.

`pcProperties` AS CHAR

Context or startup properties in the internal CHR(3) and CHR(4) delimited format.

applyContextFromServer

Applies context returned from the server after a server call.

Location: `containr.p`

Parameters: None

Returns: LOGICAL

Notes: None

assignContainedProperties

Function that returns properties in contained objects that use the returned value of containedProperties.

Location: containr.p

Parameters:

INPUT pcPropValues AS CHARACTER

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

The first entry is a header that defines which properties the remaining entries in the list applies to. The header has two possible formats:

- Comma separated list of properties when all objects are SmartDataObjects (SDOs).
- A paired semicolon lists with object type and a comma delimited list of properties. THIS is used instead of ObjectType for this container instance. For example:

`SmartBusinessObject;Prop1;SmartDataObject;propA,propB,propC`

- The remaining CHR(3) entries are a paired list of objectNames and a CHR(4)-delimited list of property values, where each CHR(4) entry corresponds to the comma separated property list for the object's Object type.
- A blank objectname indicates this container is an instance.
- ObjectNames is qualified with ':' to specify container relationships.

INPUT pcReplac AS CHARACTER

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

Returns: LOGICAL

Notes: None

assignPageProperty

Procedure that assigns a property in all objects on the CurrentPage of a SmartContainer. If an object on the page does not have a property, it is ignored without error.

Location: containr.p

Parameters:

INPUT pcProp AS CHARACTER

The property to set.

INPUT pcValue AS CHARACTER

The value to assign to that property.

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.
- Although the pcValue parameter is specified in CHARACTER form, you can use these parameters to assign values to noncharacter properties.
- All objects in a Container subscribe to initializeObject, and so on, but the paging performs the operation on subsets of those objects at a time. That is, the container does 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).
```

cancelObject

Procedure that cancels an object.

Location: `containr.p`

Parameters: None

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 container targets 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.

changePage

Procedure that views (and creates if necessary) objects on a newly selected page in a Container when `CurrentPage` is reset.

Location: `containr.p`

Parameters: None

Notes:

- The `changePage` procedure is normally used internally and called from either `selectPage` or `viewPage`. Although you can customize `changePage` to perform some application-specific task each time a page is changed, if it is called from `selectPage` (the standard methods for switching pages within a Container), the previous page will have already 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 is 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.
```

clearCache

This procedure clears cached data for a DataObject. If the cached data is still in use by a DataObject instance, the procedure only marks it as no longer valid. It is left in memory for the instances that are still using it, but no new instances will use it.

Location: cache.p

Parameters:

INPUT pcKey AS CHARACTER

The LogicalObjectName of the DataObject, or the container name and DataObject instance. An asterisk (*) clears cached data for all DataObjects.

Notes: None

confirmExit

Procedure that passes this event on to its descendents, to check whether it is OK to exit (that is, there are no unsaved changes).

Location: containr.p

Parameters:

INPUT-OUTPUT p1Cancel AS LOGICAL

If field values have been modified and not saved, it returns TRUE and the destroyObject is cancelled.

Notes: Invoked from destroyObject.

confirmOk

Procedure that verifies that unsaved changes are saved and uncommitted data is committed before allowing its container to initiate a **destroy** operation.

Location: containr.p

Parameters:

INPUT-OUTPUT p1Error AS LOGICAL

TRUE if the destroyObject is to be cancelled.

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 the container is non-virtual and non-window.

constructObject

Procedure that runs from adm-create-objects to RUN a SmartObject to establish its parent and initial property settings. The procedure first checks if a generated file exists for the object, using the LogicalObjectName property. If a generated file exists, the file is used instead.

Location: containr.p

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.

Notes: None

containedProperties

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 is 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.

Examples:

<pcQueryProps>CHR(3)<ObjectName1>CHR(3)<PropAva1ue>CHR(4)<PropBva1ue>CHR(3)<ObjectName2>CHR(3)<PropAva1ue>CHR(4)<PropBva1ue>.

Location: containr.p

Parameters:

INPUT pcQueryProps AS CHARACTER

Comma-separated lists of properties to retrieve from SmartDataObjects (SDOs). You can optionally use paired semicolon lists to query different Object types.

INPUT p1Deep AS LOGICAL

Determines the level at which a query should run. If TRUE, continue query in children of children. If FALSE, query only one level down.

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.
- If the ClientNames property is defined, the ObjectName entry in the returned list is replaced with the corresponding entry in the ClientNames list. This information is used to enable communication among clients with different container structures.

ContextandDestroy

Server-side procedure to run after new data has been requested by the client.

Location: containr.p

Parameters: None

Notes: None

createObjects

Standard procedure for running objects in a Container. This procedure runs the AppBuilder-generated procedure named adm-create-objects for compatibility with Version 8.

Location: containr.p

Parameters: None

Notes:

- This procedure runs 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 is called using createObjects.

- This can be customized to add application-specific behavior to the creation of SmartObjects in a Container. createObjects runs once for each page as it is initialized, so code in a custom version should always check the CurrentPage property to verify 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.
```

confirmCancel

Procedure that verifies that unsaved changes are cancelled and uncommitted data are undone before allowing its container to initiate a **destroy** operation.

Location: containr.p

INPUT-OUTPUT plError AS LOGICAL

TRUE if the destroyObject is to be cancelled.

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 the container is non-virtual and non-window.

deletePage

Procedure that deletes all the objects on the specified page.

Location: containr.p

Parameters:

INPUT piPageNum AS INTEGER

Number of page to be purged.

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.
```

destroyObject

Procedure that destroys a container object.

Location: containr.p

Parameters: None

Notes: None

disablePagesInFolder

Disables the specified folder pages.

Location: containr.p

Parameters:

INPUT pcPageInformation AS CHARACTER

 The pages to be disabled.

Returns: LOGICAL

Notes: None

enablePagesInFolder

Enables the specified folder pages.

Location: containr.p

Parameters:

INPUT pcPageInformation AS CHARACTER

 The pages to be enabled

Returns: LOGICAL

Notes: None

fetchContainedData

Client-side procedure that retrieves a set of data from the server for the client.

Location: containr.p

Parameters:

INPUT pcObject AS CHARACTER

 Name of the client-side query object for which data is being retrieved. If the name is:

- Unknown, then result sets are fetched for all QueryObjects in this container and all QueryObjects in all contained containers.
- Specified, then result sets are fetched from that Object down following only that data link.

Notes:

- The data links are followed when preparing and retrieving data for all objects in containers.
- The data links can go across containers but it cannot go into a container that is not a descendant of that container.

findCachelItem

This function returns TRUE if it finds a temp-table in the data cache with the passed key as identifier.

Location: `cache.p`

Parameters:

INPUT `pcKey` AS CHARACTER

The LogicalObjectName of the DataObject, or the container name and DataObject instance.

INPUT `piMaxAge` AS INTEGER

The maximum age of the cached data in seconds. (Optional)

Returns: LOGICAL

Notes: This function replaces existing cache with same key.

hidePage

Procedure that cycles through all objects on a page, hiding them.

Location: `containr.p`

Parameters: None

Notes: None

initializeObject

Procedure that 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.

Location: containr.p

Parameters: None

Notes: None

initializeVisualContainer

Procedure that translates window title and tab folder page labels and checks security for page labels.

Location: containr.p

Parameters: None

Notes: None

initPages

Procedure that initializes one or more pages, which are not yet being viewed, in order to establish links or to prepare for the pages being viewed.

Location: containr.p

Parameters:

INPUT pcPageList AS CHARACTER

A comma-separated list of page numbers.

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 detects this during the design process and automatically generates an appropriate call to `initPages` as part of `adm-create-objects`. This assure that when the dependent page is initialized, the pages it depends on are 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.

isUpdateActive

Procedure that is received from container source to check if contained objects have unsaved or uncommitted changes (including `addMode`).

Location: `containr.p`

Parameters:

INPUT-OUTPUT `p1Active` AS LOGICAL

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.

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.
```

notifyPage

Procedure that invokes the specified procedure in all objects on the CurrentPage of A SmartContainer.

Location: containr.p

Parameters:

INPUT pcProc AS CHARACTER

The internal procedure to run.

Notes:

- This procedure is necessary because paging does not work well with PUBLISH/SUBSCRIBE. For example, although all objects in a Container subscribe to initialize (and so on), the paging performs the operation on subsets of those objects.
- The notifyPage procedure uses the PageNtargets property values to run an event procedure in all the objects on the current page.

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).
```

obtainContextForServer

Function used to obtain the context properties to pass to the server.

Location: containr.p

Parameters: None

Returns: CHARACTER

Notes:

- The server must know whether or not this is the first call. Determining whether or not it is the first call can be done using either one of the following:

Client: AsHasStarted = no until a response from server

Server: serverFirstCall = no unless client passes YES

Two similar properties are provided to support different defaults on client and server.

- This function is always called before an Appserver call and is used to send context in initializeServerObject as well as in all data requests.

okObject

Procedure that saves and closes an object (OK action).

Location: `containr.p`

Parameters: None

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.

pageNTargets

Returns a comma-separated list of the objects on the specified page for this container.

Location: `containr.p`

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.

passThrough

Procedure that acts as an intermediary for dynamic links that need the pass-through mechanism to either get an event from an object outside a container to an object inside a container or to get an event from an object inside a container to an object outside a container.

Location: `containr.p`

Parameters:

INPUT `pLinkName` AS CHARACTER

The link (event) name to be passed on.

INPUT `pcArgument` AS CHARACTER

A single character-string argument.

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.

registerCacheItem

This function registers a temp-table in the data cache with the passed key as identifier.

Location: cache.p

Parameters:

INPUT pcKey AS CHARACTER

The LogicalObjectName of the DataObject, or the container name and DataObject instance.

INPUT phTable AS HANDLE

The temp-table handle to register as cached.

INPUT piTimeSpan AS INTEGER

The number of seconds for which the cached data is valid. This is typically the value of the DataObject's CacheDuration property. If 0, the data is valid only as long as it is in use. It is shared with other instances of the DataObject but is not cached. If unknown (?), the data is valid for the entire session.

INPUT piNumRecords AS INTEGER

The number of records in the temp-table. (Optional)

INPUT pcContext AS CHARACTER

Not currently used. Reserved for future development.

Returns: LOGICAL

Notes: This function replaces existing cache with same key.

removePageNTarget

Procedure that removes an object from the list of Targets on a page.

Location: containr.p

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.

Notes: Run from removeAllLinks for objects not on Page 0. Not intended to be run from application code.

resizeWindow

Procedure that respond to a resize event from user event or container targets.

Location: containr.p

Parameters: None

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.

selectPage

Procedure that 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.

Location: containr.p

Parameters:

INPUT piPageNum AS INTEGER

Notes:

- This procedure switches from one page on a single frame to another page so that objects on the previous page are hidden and objects on the new page are viewed. If the new page is a separate SmartWindow, in most cases you should use the viewPage procedure to view the objects on the new page without hiding the current page.
- The selectPage procedure runs when a user presses a tab on the SmartFolder. It can also 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.
```

targetPage

Returns the ADM page number associated with some object.

Location: containr.p

Parameters:

INPUT phObject AS HANDLE

 The object of interest

Returns: INTEGER

Notes: None

toolbar

Procedure that is a generic event handler for toolbar events.

Location: containr.p

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.

Notes: None

updateActive

Procedure that is published from `ContainerTargets` when they change state as a result of, for example, `setDataModified`, `setNewRecord`, or `setRowObjectState`.

Location: `containr.p`

Parameters:

INPUT `pActive` AS LOGICAL

TRUE if some update is known active and `ContainerTargets` need not be surveyed

Notes: This is part of the logic to make the `UpdateActive` property reflect the containers state.

viewObject

Procedure this is 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.

Location: `containr.p`

Parameters: None

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.

viewPage

Procedure that views a new page without hiding the current page. You should run this from application code when you want a user to view a new page that is a separate SmartWindow. viewPage runs changePage to view (and if necessary create) the new SmartWindow, but does not hide the objects on the current page, since they are in a separate window that can be viewed at the same time.

Location: `containr.p`

Parameters:

INPUT piPageNum AS INTEGER

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.

```

Methods for SmartBusiness container objects

The following section describes the methods for SmartBusinessObjects (SBOs). The SBO integrates multiple SmartDataObjects and is a special purpose organizer object that is part of the Container class. SBOs provide a single point of contact for other objects, and allow you to synchronize updates on multiple SmartDataObjects in a single server-side transaction.

addDataTarget

Procedure that 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.

Location: sbo.p

Parameters:

INPUT phTarget AS HANDLE

The target handle to be added.

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.

addNavigationSource

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

Location: `sbo.p`

Parameters:

INPUT `phSource` AS HANDLE

The handle of the source.

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

addQueryWhere

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

Location: `sbo.p`

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.

addRow

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

Location: sbo.p

Parameters:

INPUT pcViewColList AS CHARACTER

Returns: CHARACTER

Notes: None

appendContainedObjects

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

Location: sbo.p

Parameters:

INPUT-OUTPUT pcObjects AS CHARACTER

INPUT phObject AS HANDLE

Returns: LOGICAL PRIVATE

Notes: Private function.

assignCurrentMappedObject

Current contained Data Object for Navigation or other access by the caller.

Location: sbo.p

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.

assignMaxGuess

Procedure that identifies the MaxGuess event from a contained SDO that was passed to the appropriate Data-Target.

Location: sbo.p

Parameters:

INPUT piMaxGuess AS INTEGER

Notes: None

assignQuerySelection

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

Location: sbo.p

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 name is replaced with RowObject when the columns are passed on to the SmartDataObject.

cancelNew

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

Location: sbo.p

Parameters: None

Notes: None

cancelRow

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

Location: sbo.p

Parameters: None

Returns: CHARACTER

Notes: See the entry for data.p.

canNavigate

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.

Location: sbo.p

Parameters: None

Returns: LOGICAL

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.

colValues

Locates requested columns in contained Data Objects and assembles a list of their values. SmartBusinessObject version of the similar SmartDataObject function.

Location: sbo.p

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 is the RowIdent from that SmartDataObject (its tt rowid + db rowids). Otherwise, a list of all the tt rowids **without** their db rowids is returned. This allows submitRow() to send the correct rowid on to each contained SmartDataObject on update.columnWidth (sbo.p).

commitData

Procedure that calls undoTransaction to clean up temp tables after the commit operation finishes.

Location: sbo.p

Parameters:

OUTPUT pcError AS CHAR

commitTransaction

Client-side event procedure that receives the Commit event, collects the updates from contained SmartDataObjects, and passes the update to the server.

Location: sbo.p

Parameters: None

Notes: None

confirmContinue

If called with a value of FALSE, asks all contained SDOs whether they have changes pending.

Location: sbo.p

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.).

copyRow

Passes the column list on to the contained DataObject that manages that data.
SmartBusinessObject version of this function.

Location: sbo.p

Parameters:

INPUT pcViewColList AS CHARACTER

Returns: CHARACTER

Notes: Qualify all columns or no columns.

currentMappedObject

Returns the object name that is currently mapped to the calling routine.

Location: sbo.p

Parameters:

INPUT phRequester AS HANDLE

Returns: CHARACTER

Notes: The value is derived from the ObjectMapping property in the SmartDataObject.

dataAvailable

Procedure that 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.

Location: sbo.p

Parameters:

INPUT pcRelative AS CHARACTER

Flag indicating the state of the record.

Notes: See the description of the SmartDataObject return for more information.

dataObjectHandle

Returns the handle of the ObjectName (logical name) of a contained SmartDataObject.

Location: sbo.p

Parameters:

INPUT pcObjectName AS CHARACTER

Returns: HANDLE

Notes: None

deleteComplete

Procedure that receives the deleteComplete event from a contained SDO and passes it on to the appropriate DATA-TARGET.

Location: sbo.p

Parameters: None

Notes: None

deleteRow

Passes the rowident on to the contained DataObject that manages that data.
SmartBusinessObject version of this function.

Location: sbo.p

Parameters:

INPUT pcRowIdent AS CHARACTER

Returns: LOGICAL

Notes: None

destroyServerObject

Procedure that fetches context from the server-side procedure.

Location: sbo.p

Parameters: None

Notes: unbindServer is the public interface to this procedure.

endClientDataRequest

Procedure that contains logic to retrieve data properties from the Appserver after a data request. Both queries and commits are considered as data requests.

Location: sbo.p

Parameters: None

Notes: The purpose of this function is to encapsulate the logic for stateless and state-aware requests in one call.

fetchBatch

Procedure that returns the next batch of rows to a browse object, communicating with its SmartDataObject.

Location: sbo.p

Parameters:

INPUT p1Forwards AS LOGICAL

TRUE if moving forward in the database, FALSE if backward.

Notes: None

fetchContainedData

Procedure that retrieves a set of data from the server, given a where-clause for (currently) the top-level SmartDataObject. Client-side procedure.

Location: sbo.p

Parameters:

INPUT pcObject AS CHARACTER

If specified, then fetch result sets from that Object down only.

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

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.

Location: sbo.p

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

Notes: If pcObject AS CHARACTER is specified, then this procedure only fetches result sets from that Object on down. Intended for internal use. Called from clientSendRows in the contained SDO.

fetchDOProperties

Procedure that retrieves any properties from the server-side SmartBusinessObject and its SmartDataObjects, and sets them in the contained SmartDataObjects on the client.

Location: sbo.p

Parameters: None

Notes: Currently retrieves the OpenQuery property, which is set only on the server but needed on both sides.

fetchFirst

Procedure that 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.

Location: sbo.p

Parameters: None

Notes: Uses the MasterDataObject by default.

fetchLast

Procedure that 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.

Location: sbo.p

Parameters: None

Notes: Uses the MasterDataObject by default.

fetchNext

Procedure that 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.

Location: sbo.p

Parameters: None

Notes: Uses the MasterDataObject by default.

initDataObjectOrdering

Initializes the mapping of the AppBuilder-generated order of Upd tables to the developer-defined update order.

Location: sbo.i

Parameters: None

Returns: CHARACTER

Notes: This is used in commitTransaction and serverCommitTransaction to order the table parameters properly.

fetchPrev

Procedure that 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.

Location: sbo.p

Parameters: None

Notes: Uses the MasterDataObject by default.

findRowWhere

Finds a row and repositions to that row.

Location: sbo.p

Parameters:

INPUT pcColumns AS CHARACTER

For a SmartBusinessObject (SBO), 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),

For a SmartDataObjects (SDO), column names (comma separated); fieldname of a table in the query in the form of RowObject.FLDNM.

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:

- This method resolves the row reposition on the server. As a result, the SDO can no longer determine whether the row position is invalid until after the request has been executed. However, if the RebuildOnRepos property is set to TRUE, the Temp-table is emptied *before* the request.
- The current behavior for a FIND that does not find anything when RebuildOnRepos is TRUE is to read the current batch again.
- The logic is in the query.p super procedure.

initializeObject

Procedure that initializes objects of class SBO. Sets the MasterDataObject property to the leading SmartDataObject.

Location: sbo.p

Parameters: None

Notes: Later this procedure must also establish the AppServer connection and other tasks.

initializeServerObject

Procedure that sets context and initializes the server object.

Location: sbo.p

Parameters: None

Notes: None

isUpdatePending

Procedure that is 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.

Location: sbo.p

Parameters:

INPUT-OUTPUT p1Update AS LOGICAL

Returns TRUE and stops the publication if update is pending.

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.

newDataObjectRow

Prepares one or more contained SDOs to add or copy (create) a new record.

Location: sbo.p

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.

openQuery

A wrapper used by fetchContainedData(), allows you to use the SmartDataObject calling sequence.

Location: sbo.p

Parameters: None

Returns: LOGICAL

Notes: None

postCreateObjects

Procedure that 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.

Location: sbo.p

Parameters: None

Notes: None

prepareErrorsForReturn

Procedure that 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.

Location: sbo.p

Parameters:

INPUT pcReturnValue AS CHARACTER

INPUT pcASDivision AS CHARACTER

INPUT-OUTPUT pcMessages AS CHARACTER

Notes: Invoked internally from serverCommitTransaction().

prepareQueriesForFetch

Procedure that prepares queries in SmartDataObjects for a fetch of data from the server.

Location: sbo.p

Parameters:

INPUT pcObjectName AS CHARACTER

INPUT pcOptions AS CHARACTER

OUTPUT pocQueries AS CHARACTER

OUTPUT poctempTables AS CHARACTER

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.

queryPosition

Procedure that receives the queryPosition event from a contained SmartDataObject and passes it on to the appropriate Navigation-Source or other object.

Location: sbo.p

Parameters:

INPUT pcPosition AS CHARACTER

Notes: None

refreshBrowse

Procedure that receives the refreshBrowse event from a contained SDO and passes it on to the appropriate DataTarget.

Location: sbo.p

Parameters:

INPUT pcPosition AS CHARACTER

Notes: None

registerLinkedObjects

Procedure that registers objects in the ObjectMapping and other properties. This currently applies to navigationSources and DataTargets. Currently we assume that UpdateSources are DataTargets also.

Location: sbo.p

Parameters: None

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.

registerObject

A general purpose register event procedure 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.)

Location: sbo.p

Parameters: None

Notes: None

remoteCommitTransaction

Server-side version of CommitTransaction that receives all RowObjUpd table updates and passes them on to server-side SmartDataObjects (SDOs).

Location: sbo.i

Parameters:

INPUT-OUTPUT pccontext AS CHAR

RowObjUpd table references. This parameter can include up to twenty RowObjUpd table references.

OUTPUT pcMessages AS CHARACTER

The generated error messages.

OUTPUT pcUndoIds AS CHARACTER

Rowids of rows that generated error messages.

NOTE: Unused tables use the placeholder TABLE-HANDLE.

remoteFetchContainedData

Server-side procedure that prepares and opens a query and returns all the resulting data to the client.

Location: sbo.p

Parameters:

INPUT-OUTPUT pcContext AS CHARACTER

Input context from and output context to client.

pcQueries AS CHARACTER

CHR(1)-delimited-list of QueryString properties of the SmartDataObjects (SDOs).

pocMessages

Error messages generated.

remoteSendRows

A stateless version of sendRows that does no processing but runs sendRows to pass all parameters except for the context and returns the RowObject table as an output parameter to the caller that has the new batch of records created in sendRows.

Location: sbo.p

Parameters:

INPUT-OUTPUT pcContext

CHR(3) separated list of propCHR(4)value pairs. INPUT is the current context and OUPUTPUT is the new context. The INPUT and OUTPUT can have different properties.

INPUT piStartRow

The RowNum value of the record to start the batch to return. Typically piStartRow is a flag with an unknown value (?) that indicates pcRowIdent should be used instead of piStartRow.

INPUT pcRowIdent

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

Determines whether serverSendRows should start on the next record instead of what is indicated by piStartRow or piRowIdent. If TRUE, serverSendRows should start on the next record instead of what is indicated by piStartRow or piRowIdent

INPUT `piRowsToReturn`

The number of rows in a batch.

OUTPUT `piRowsReturned`

The actual number of rows returned. This number is either the same as `piRowsToReturn` or less if there are not enough records to fill the batch.

OUTPUT `pcMessages`

Used for error messages.

Notes:

- 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`.
- The object should ***only*** be started persistently before this is called and ***not*** initialized because `initializeObject` is run ***after*** the context has been set.
- The caller is responsible for destroying the object.
- For more details, see `synchronizeProperties` and `genContext`.

removeQuerySelection

Separates the Columns by SmartDataObject and passes columns and operators on to the appropriate SmartDataObjects. SmartBusinessObject version of this where-clause function.

Location: `sbo.p`

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 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 SDO ObjectName as TableName. This name is replaced with RowObject when columns are passed on to the SDO.

repositionRowObject

Passes the rowident on to the contained DataObject that manages that data. SBO version of the routine.

Location: `sbo.p`

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

resetQuery

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.

Location: sbo.p

Parameters:

INPUT pcObject AS CHARACTER

SmartDataObject ObjectName.

Returns: LOGICAL

Notes: This function is here because QueryWhere is not supported for SmartBusinessObjects, and running QueryWhere("") would be the way to do it otherwise.

restartServerObject

A procedure that shuts down a SmartBusinessObject after each use and then restarts it when the SmartBusinessObject is split and running stateless on an AppServer. restartServerObject is run on the client to restart the SmartBusinessObject on the server.

Location: sbo.p

Parameters: None

Notes: This override is for error handling to show error message and return adm-error.

serverCommitTransaction

Server-side version of CommitTransaction. This procedure receives all RowObjUpd table updates and passes them on to server-side SDOs.

Location: sbo.i

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 SmartDataObject. 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.

Notes: None

serverContainedSendRows

A procedure that receives the SmartDataObject ObjectName and runs SendRows in that, returning the RowObject table. Server-side SmartBusinessObject version of serverSendRows.

Location: sbo.p

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 p1Next 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 is either 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.

Notes: None

serverFetchContainedData

Procedure that prepares and opens a query and returns all the resulting data to the client side. Server-side procedure.

Location: sbo.p

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.

Notes: None

serverFetchContainedRows

Procedure that prepares and opens a query on the server side and returns all resulting data to the client side.

Location: sbo.p

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.

Notes: None

serverFetchDOProperties

A procedure that runs at startup to return property values needed on the client. Server-side procedure.

Location: sbo.p

Parameters:

OUTPUT pcPropList AS CHARACTER

Notes: Currently returns the OpenQuery property of each SmartDataObject.

setPropertyList

A list of properties taken from a CHR(3)-delimited list of “prop CHR(4) value” pairs.

Location: sbo.p

Parameters: None

startServerObject

When a SmartBusinessObject is split and running statelessly on an AppServer, the startServerObject procedure is run on the client to start the SmartBusinessObject on the server.

Location: sbo.p

Parameters: None

Notes: This override is for error handling to show error message and returns **adm-error**.

submitRow

Accepts a list of changed values for a row and passes them on to the SmartDataObjects from which they came.

Notes: sbo.p

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

undoTransaction

Procedure that passes the undoTransaction event on to each contained SmartDataObject that has any uncommitted changes.

Location: sbo.p

Parameters: None

Notes: None

updateState

Procedure that republishes any updateState event messages received from a Data-Target, to get them, for example, to Navigation Panel/Toolbar.

Location: sbo.p

Parameters:

INPUT pcState AS CHARACTER

Notes: None

Methods for TreeView controller container objects

The following section describes the methods for TreeView controller objects.

initializeObject

TreeViewController initialization procedure.

Location: tvcontroller.p

Parameters: None

Notes: None

postCreateObjects

Procedure that 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 TreeView.

Location: tvcontr.p

Parameters: None

Notes: None

showTVCErrors

A procedure that runs showMessages in the Session Manager to display a TreeView controller error message.

Location: tvcontroller.p

Parameters:

INPUT pcMessage AS CHARACTER

The message to be displayed

Notes: None

updateState

A procedure that disables the Treeview during user editing and re-enables on completion.

Location: tvcontroller.p

Parameters:

INPUT pcState AS CHARACTER

The new state.

Notes: None

Container object properties

Container Object properties provide information about container objects and their classes. This information can include whether an object is enabled, the contents of the object and so on. You can read property values and in many instances you can change property values. To read a property value, you use a **get** function, and to change a property value, you use a **set** function.

These functions conform to the following conventions:

- **get** — Uses the form `get $\textit{propnam}$` and returns the current value of the property

NOTE: This function accepts no arguments.

- **set** — Uses the form `set $\textit{propname}$` . The set function accepts a single argument—the new value for the property—and returns TRUE/FALSE depending on whether the value change succeeds.

For more information about getting and setting property values, see [Chapter 1, “ADM2 SmartObject API Reference”](#) in this guide and the *Progress Dynamics Programming Handbook*.

BlockDataAvailable

Controls whether or not DataAvailable messages from contained SDOs are to be ignored and not republished. If TRUE, DataAvailable messages are to be ignored and not republished.

Data Type: LOGICAL

Notes: Read and Write

CallerObject

Value of CallerObject.

Data Type: HANDLE

Notes: Read and Write

CallerProcedure

Value of CallerProcedure.

Data Type: HANDLE

Notes: Read and Write

CallerWindow

Value of CallerWindow.

Data Type: HANDLE

Notes: Read and Write

CascadeOnBrowse

Determines whether data is retrieved from a dependent SDO if the parent SDO has more than one row in its current result set. If TRUE (the default), data is retrieved for the first row in the parent result set, otherwise not.

Data Type: LOGICAL

Notes: Read and Write

CommitSource

Used for pass-through for regular containers, but also inherited by the SBO, which uses it generally.

Data Type: HANDLE

Notes: Read and Write

CommitSourceEvents

Represents the list of events to be subscribed to in the Commit Panel or other Commit-Source.

Data Type: CHARACTER

Notes:

- 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.

Notes: Read and Write

CommitTarget

List in character format the handles of this object's Commit-Targets.

Data Type: CHARACTER

Notes: Read and Write

CommitTargetEvents

List of events to be subscribed to in the Commit Panel or other Commit-Target.

Data Type: CHARACTER

Notes: Read and Write

ContainedDataColumns

Delimited list of all the DataColumns of all the Data Objects in this SBO.

Data Type: CHARACTER

Notes: Read and Write

ContainedDataObjects

List of the handles of the Data Objects contained in this container object.

Data Type: CHARACTER

Notes:

- Read and Write
- The container class uses this property to keep track of the data objects for the stateless server side
- The SmartBusinessObject (SBO) class uses this property on both the client and server for most logic and at design time to get names and column lists from the individual data objects.

ContainerTarget

List of the handles of the object's contained objects.

Data Type: CHARACTER

Notes: Read and Write

ContainerTargetEvents

Comma-separated list of the events this object wants to subscribe to in its ContainerTarget.

Data Type: CHARACTER

Notes: Read only

ContextAndInitialize

Resets context and initializes this server-side object. Called from a stateless client before a request.

Data Type: CHARACTER

Notes: Write only

CurrentPage

Current page number of the Container.

Data Type: INTEGER

Notes: Read and Write

DataHandle

Handle of the RowObject query for the SBO.

Data Type: HANDLE

Notes: Read only

DataObjectNames

Ordered list of ObjectNames of contained SDOs.

Data Type: CHARACTER

Notes:

- Read and Write
- 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.
- Used to 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 this list no longer matches the list of contained SDOs, then it is blank and a default list is created again.

DataObjectOrdering

Mapping of the order of Update Tables as generated by AppBuilder to the developer-defined update order.

Data Type: CHARACTER

Notes: Write only

DataQueryBrowsed

DataQueryBrowsed value after mapping the requesting Browser (or other such client object) to the SDO whose query it is browsing.

Data Type: LOGICAL

Notes: Read and Write

DynamicSDOProcedure

Name of the dynamic SmartDataObject (SDO) procedure. That is `adm/dyndata.w` by default, but it can be modified if the dynamic SDO is customized.

Data Type: CHARACTER

Notes: Read and Write

FetchOnOpen

Provides a consistent interface for all Data Objects.

Data Type: CHARACTER

Notes: Read and Write

FilterSource

Represents the Filter Source for pass-through support.

Data Type: HANDLE

Notes: Read and Write

InitialPageList

Comma-delimited list of pages to construct at startup, or * to indicate all pages must be initialized at startup.

Data Type: CHARACTER

Notes: Read and Write

InMessageTarget

Value of InMessageTarget.

Data Type: HANDLE

Notes: Write only

InstanceNames

Property that provides an ordered list of ObjectNames of ContainerTargets.

Data Type: CHARACTER

Notes:

- Read and Write
- This property is used to enforce unique instance names in the container and is updated in `constructObject` and `destroyObject` along with the container link.
- This property allows `containedProperties` and `assignContainedProperties` to work together.

LastCommitErrorKeys

Property that provides information about records that failed during the last data commit. For:

- **SmartDataObjects (SDOs)** — A comma-delimited list of the key values of the records that failed to be committed. The `KeyFields` property of the SDO holds the key field names.
- **SmartBusinessObjects (SBOs)** — A semicolon-delimited list of the values of each individual contained SDO that has failed records.

Data Type: CHARACTER

Notes:

- Read and Write
- A blank indicates that the last commit was successful.

LastCommitErrorType

Property that identifies the type of error encountered the last time data was committed:

- **Blank** — The last commit was successful.
- **Unknown** — A commit was not attempted after run.
- **Conflict** — A locking conflict occurred.
- **Error** — An unspecified error occurred.

Data Type: CHARACTER

Notes:

- Read and Write
- Currently used to identify a Conflict error when using the `UpdateData` procedure. See [Update Data](#) for additional information.

MasterDataObject

Handle of the SDO that has no data source of its own and is the parent to other SDOs.

Data Type: HANDLE

Notes: Read only

MultiInstanceActivated

Value of MultiInstanceActivated.

Data Type: LOGICAL

Notes: Read and Write

MultiInstanceSupported

Value of MultiInstanceSupported.

Data Type: LOGICAL

Notes: Read and Write

NavigationSource

Comma-separated list of strings for Navigation sources. Used for pass-thru for regular containers, but also inherited by the SBO that actually uses it.

Data Type: CHARACTER

Notes: Read and Write

NavigationSourceEvents

List of events to be subscribed to in the Navigation Panel or other Navigation-Source.

Data Type: CHARACTER

Notes: Read and Write

NavigationTarget

Value of NavigationTarget.

Data Type: HANDLE

Notes: Read and Write

ObjectMapping

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.

Data Type: CHARACTER

Notes: Read and Write

OutMessageTarget

Out message target for Pass-through support.

Data Type: HANDLE

Notes: Read and Write

PageNTarget

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.

Data Type: CHARACTER

Notes: Read and Write

PageSource

Handle of the object's folder, if any.

Data Type: HANDLE

Notes: Read and Write

PrimarySdoTarget

Value of PrimarySdoTarget.

Data Type: HANDLE

Notes: Read and Write

QueryPosition

For SBOs, indicates whether the requester is the target-procedure or the source-procedure, and whether the MasterDataObject is involved, and identifies the SDO to which the caller is mapped.

Data Type: CHARACTER

Notes: Read and Write

RouterTarget

Value of RouterTarget.

Data Type:

Notes: Read and Write

RowObjectState

Signals whether there are uncommitted updates in the object. Valid return values are **NoUpdates** and **RowUpdated**.

Data Type: CHARACTER

Notes: Read and Write

RunDataLogicProxy

Property that provides user control over which Data Logic Procedure (DLP) file to run.

Data Type: LOGICAL

Notes:

- Read and Write.
- The default value is Unknown. When the default value is used, which DLP file to run is determined by a search process that depends on the client type and whether the file can be found. The following list describes the search process for a DLP file with the name *myDLP.p*:
 - For WebClients, run *myDLP_c1.p* and automatically download the DLP file if necessary.

- For GUI clients, determine whether **any** databases are connected. If yes, search for *myDLP.r*. If found, verify that **all required** databases are connected and if they are connected, run *myDLP.p*. If *myDLP.p* cannot be found and run using this search criteria, then do the following:

Search for *myDLP_c1.r*. If *myDLP_c1.r* is found, then run *myDLP_c1.p*. If *myDLP_c1.r* is not found, then search for *myDLP_c1.p*. If *myDLP_c1.p* is found, run the file. If *myDLP_c1.p* is not found, a cannot find *myDLP_c1.p* error message displays.

- For both GUI clients and WebClient, if the selected DLP file fails to run, the SmartDataObject (SDO) does not start and an error message displays.
- If an appropriate DLP file in **not** found, an error message displays under the following conditions:
 - RunDataLogicProxy=TRUE.
 - Session:ClientType=WebClient.
 - All required databases are not connected.
- A TRUE value forces the DLP proxy to run so that it can be downloaded automatically in a WebClient environment.
- A FALSE value starts the data object **without** the DLP.
- For related information, see the “[RunDOOptions](#)” property.

RunDOOptions

Comma-separated list of options that determine how Data Objects are run from `constructObject`. The available options are:

- **dynamicOnly** — Run the Dynamics client proxy SDO specified in the `DynamicSDOProcedure` property. By default, this is `adm2/dyndata.w`. When you select this option, no other version of the data object will be run.
- **sourceSearch** — Search for and run the actual source object. If it is not available or the required databases are not connected, use the `clientOnly` option.
- **clientOnly** — Do not locate or run the actual object file (*sdo-name.w*). Instead, locate the static client file (`*_cl.w`) and run it. If this file is not available, run the Dynamics client proxy.
- **StaticClientOnly** — This option is similar to the `clientOnly` option, except that the Dynamics client proxy (`dyndata.w`) is *not* run if the static client (`*_cl.w`) is not available. This option takes precedence over the **dynamicOnly** option when there is a conflict.

Data Type: CHARACTER

Notes:

- Read and Write
- The `StaticClientOnly` option is useful in a `WebClient` session when you do not want to run the Dynamics client proxy and the static client does not reside on the client. Under those conditions, selecting the `StaticClientOnly` option forces an automatic `WebClient` download of the static client (`*_cl`).
- If `dynamicOnly`, `clientOnly`, or both of these options are specified, the Data Object must have an `AppServer` partition defined. If an `AppServer` partition is not defined, and `dynamicOnly`, `clientOnly`, or both options are specified, you will experience the same errors that occur when the Data Object is running on an `AppServer` that has not been started.
- To run source code when no `rcode` is found, you must customized `containr.p` by setting `RunDOOptions` to **sourceSearch** in `constructObject`.

SdoForeignFields

Value of SdoForeignFields.

Data Type: CHARACTER

Notes: Read and Write

TopOnly

Indicates whether or not to get the window top only (toggle).

Data Type: LOGICAL

Notes: Read and Write

UpdateActive

Indicates whether any contained objects have active updates. TRUE if ANY of the contained objects have active updates.

Data Type: LOGICAL

Notes: Read and Write

UpdateSource

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.

Handle of the object's update-source.

Data Type: CHARACTER

Notes: Read and Write

WaitForObject

Handle of the object (most likely a SmartConsumer) in the container that contains a wait-for that needs to be started with startWaitFor.

Data Type: HANDLE

Notes:

WindowFrameHandle

Handle of the optional Frame widget of a Window container.

Data Type: HANDLE

Notes:

- Read and Write
- This property **only identifies** the frame of a window container. This is not the same as ContainerHandle, which is the widget handle of the container in all object. In most cases the ContainerHandle is also the Frame handle 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, and so on.

Column properties for container objects

There are a number of column properties available for which you can obtain and set (write) field values. All of these properties can be read and some of them can be set. To read and set column properties, you use the following prefixes with the specific column property:

- **Column** — Use to read the value of a specific column property. For example, if you want to read the value of the ColumnLabel property, you would specify **ColumnColumnLabel**. This would return the label of the column you specify. To obtain the data type for a specific column property, you would specify **ColumnDataType**. This would return the data type of the specified column.
- **Assign** — Use to set the value of the specified column property.

NOTE: For Container objects, there are no column properties for which you can assign the value.

For additional information, see the *Progress Dynamics Programming Handbook*.

Table 4–1 lists the column properties for container objects, provides a brief description of each property, indicates whether the property can be read or set (write), and lists the data type for each property.

Table 4–1: Column properties for container objects

(1 of 4)

Column property	Description	Read	Write	Data type
ColumnLabel	Label of the specified column.	Yes	No	CHARACTER
DataType	Data type of the specified column.	Yes	No	CHARACTER
DbColumn	Qualified database name (DatabaseName.TableName.FieldName) mapped to the RowObject column identified by pcColumn.	Yes	No	CHARACTER
Extent	Extent of the specified RowObject column. SmartBusinessObject version of the SmartDataObject.	Yes	No	INTEGER
Format	Format of the specified RowObject column. SmartBusinessObject version of the SmartDataObject.	Yes	No	CHARACTER
Help	Help text for the specified RowObject column. SmartBusinessObject version of the SmartDataObject.	Yes	No	CHARACTER
Initial	Initial value for a specified field as a character string with the field FORMAT applied. SmartBusinessObject version of the SmartDataObject.	Yes	No	CHARACTER
Label	Label for the specified column.	Yes	No	CHARACTER

Table 4–1: Column properties for container objects*(2 of 4)*

Column property	Description	Read	Write	Data type
LabelDefault	Design-time function. Returns TRUE if the label for the specified Filter-Target column is overridden.	Yes	No	CHARACTER
Mandatory	SmartBusinessObject version of the SmartDataObject, indicates whether or not a specified RowObject field is mandatory. TRUE if mandatory, FALSE if not mandatory.	Yes	No	LOGICAL
Modified	SmartBusinessObject version of the SmartDataObject, indicates whether or not a specified RowObject field has been modified. TRUE if modified, FALSE if not modified.	Yes	No	LOGICAL
ObjectHandle	Procedure handle of the first SmartDataObject in the specified SmartBusinessObject that has the specified SmartDataObject column name.	Yes	No	HANDLE
PrivateData	Private Data property of the specified RowObject field for the SmartBusinessObject version of the SmartDataObject.	Yes	No	CHARACTER

Table 4–1: Column properties for container objects*(3 of 4)*

Column property	Description	Read	Write	Data type
QuerySelection	SmartBusinessObject version of the SmartDataObject, this is a CHR(1)-separated string with all operators and values, but no field names, that have been added to the Query for this column.	Yes	No	CHARACTER
ReadOnly	SmartBusinessObject version of the SmartDataObject, this property identifies whether or not the specified column is Read Only in the owning SmartDataObject. TRUE, if the column is Read Only and FALSE, if not Read Only.	Yes	No	LOGICAL
StringValue	SmartBusinessObject version of the SmartDataObject, this property is the String Value of the specified column in the SmartDataObject RowObject buffer.	Yes	No	CHARACTER
Table	SmartBusinessObject version of the SmartDataObject, this property is the database table name of the specified column.	Yes	No	CHARACTER
ValExp	SmartBusinessObject version of the SmartDataObject, this property is the validation expression for the specified column in a RowObject temp-table.	Yes	No	CHARACTER

Table 4–1: Column properties for container objects (4 of 4)

Column property	Description	Read	Write	Data type
ValMsg	SmartBusinessObject version of the SmartDataObject, this property is the validation message for the specified column in a Row-Object temp-table.	Yes	No	CHARACTER
Value	SmartBusinessObject version of the SmartDataObject, this property is the raw (unformatted) character value of the specified field in a RowObject temp-table.	Yes	No	CHARACTER
Width	SmartBusinessObject version of the SmartDataObject, this property is the width in character units of the specified column.	Yes	No	DECIMAL

Query Objects and Their Methods and Properties

This chapter lists and describes the methods (internal procedures and functions) and properties used for SmartQuery Objects. Refer to [Figure 1–1](#) to see the inheritance hierarchy for each object class.

NOTE: For information specific to the WebSpeed environment (see the [“Alphabetical Listing Of WebSpeed-specific API Routines”](#)).

Base methods for query objects

The following section lists and describes the base methods for query objects.

addNotFoundMessage

Function that adds an error message for a record not found based on keys and values using the same format as findRowWhere and other query functions.

Location: `query.p`

Parameters:

INPUT `pcFields` AS CHAR

Comma separated list of column names.

INPUT `pcValues` AS CHAR

CHR(1) separated list of corresponding values.

Notes: The message is for unique finds.

addQueryWhere

Function that adds string-expressions to the query's WHERE clause and stores the result in the QueryString property. The function returns TRUE if successful, FA LSE if an appropriate buffer name for the WHERE-clause cannot be located.

Location: `query.p`

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 so that it can be called multiple times. This design lets you add multiple phrases to the where-clause before the full where-clause is used to reopen the query.

assignDBRow

Procedure that modifies values copied from the RowObject row to the database records (which for an Update have already been retrieved and locked).

Location: query.p

Parameters:

INPUT phRowObjUpd AS HANDLE

Handle of the buffer from which to copy.

Notes:

- The SmartDatatObject determines which fields to save to the database tables based on a comparison of the before-image and the changed record.
- The RowObjUpd.ChangedFields that was used for this purpose is now obsolete.
- 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.
- If the RowMod field is **A** for Add or **C** for Copy, the procedure creates the database records first and then does the assign.

assignQuerySelection

Selection criteria for a query. This is used by the column/value pairs of the corresponding buffer's where-clause.

Location: `query.p`

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:

- Each buffer's expression is always enclosed in parentheses.
- This procedure is designed to run on the client so that it can be called multiple times. This design lets you build up the query's WHERE clause by storing intermediate results in the QueryString property before it is finally used in a Query-Prepare method.
- `openQuery` helps prepare the QueryString property.
- The QueryColumns property ensures that each column and operator is added only once to the QueryString. The property also stores the offset and length of the corresponding values.

batchServices

A procedure that groups a sequence of SmartDataObject service requests into a single request and thereby minimizes network messaging and improves performance.

Location: query.p

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.

Notes: 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 is 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.

bufferCopyDBToRO

Procedure that 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 ensures that the values are copied properly.

Location: `query.p`

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.

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 is found (for example, `pcAssigns` = “ROfld1,DBfld[1]”).

ColumnPhysicalColumn

Function that returns the qualified physical name ([DB.]TBL.FLDNM) mapped to the RowObject column specified in `pcColumn`.

Location: `query.p`

Parameters:

INPUT `pcColumn` AS CHAR

Rowobject name to look up.

Returns: CHARACTER

Notes: None.

ColumnPhysicalTable

Function that returns [dbname.]table of a rowobject or database column.

Location: query.p

Parameters:

INPUT pcColumn AS CHAR

Database fieldname. Can be in the form of: DB.TBL.FLDNM, TBL.FLDNM or FLDNM.
If not qualified, the FIRST reference in query is used)

Returns: CHARACTER

Notes:

- Used to ensure and fix a column reference according to the query's use of database qualification. See dbColumnHandle for additional information.
- Supports a pcColumn specified with brackets.

closeQuery

Closes the database query.

Location: query.p

Parameters: None

Returns: LOGICAL

Notes: If closeQuery is executed on the client side of a split SmartDataObject when the database query only exists on the server side, nothing is closed.

colValues

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.

Location: `query.p`

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.

confirmCommit

Procedure that 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`) however, offers the user the opportunity to save or cancel a record in order to be able to commit.

Location: `query.p`

Parameters:

INPUT-OUTPUT `pioCancel` AS LOGICAL

Returns TRUE if the transaction should be cancelled.

Notes: None

confirmContinue

Asks all data-targets whether they have changes pending.

Location: `query.p`

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.)

confirmUndo

Procedure that, if called with a `FALSE` argument, publishes `confirmUndo` all data-targets to determine whether any are Modified or in AddMode. Returns `TRUE` if the undo should be cancelled. The visual objects (`visual.p`) warns the user that unsaved changes will be cancelled.

Location: `query.p`

Parameters:

INPUT-OUTPUT `pioCancel` AS LOGICAL

Returns `TRUE` if the data-targets agree that undo can proceed.

Notes: If called with a `TRUE` argument, the routine does nothing.

dataAvailable

Event procedure that generates a dependent query dynamically based on the ForeignFields property. This event occurs when dataAvailable is published by a data source because the data source has been repositioned to a different row in its query and this has an impact on dependent objects.

Location: `query.p`

Parameters:

INPUT `pcRelative` AS CHARACTER

Provides information about a newly available record. Valid entries 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 to other target procedures so that it appears as though the change occurred in this procedure.
- **RESET** — Resets the status and foreign fields, and refreshes visual objects and panels for all objects that are part of the data link. This option provides more functionality than **SAME** and less functionality than **DIFFERENT**. Use this option when you want to send notification about a change in the RowObject record without having to reopen all the dependent queries.
- **DIFFERENT** — Values for Foreign Fields should be reapplied.
- **FIRST, NEXT, PREV, LAST** — Treated the same as **DIFFERENT** in this version of dataAvailable.
- **TRANSFER** — Joins the child data object to the current record in a parent data object. As a result, the call only needs to be performed in the parent data object inside of a container.

Notes: This version of data Available is for SmartDataObjects that depend on another SmartDataObject. The code is different from that for a Viewer. If there are no Foreign Fields, then this procedure is run when a target is repositioned (usually a SmartDataBrowser). In this case, just the event is passed on to other targets.

dbColumnNameDataName

Returns the RowObject field name of a database field name.

Location: query.p

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

Returns the handle of a database column.

Location: query.p

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 columnNameDataType, ColumnValMsg, and ColumnTable.

defineDataObject

Function used to define the following Dynamics data object properties:

- Tables
- BaseQuery
- DataColumns
- DataColumnsByTable
- UpdatableColumnsByTable
- AssignList

Location: query.p

Parameters:

INPUT pcTableList AS CHARACTER

Comma separated list of tables. If the table names are database qualified then the column list also needs to be database qualified.

INPUT pcBaseQuery AS CHARACTER

Query used for the dynamic SmartDataObject.

INPUT pcColumnList AS CHARACTER

Comma separated list of qualified columns in the format:

TableName.ColumnName[.RemanedColmnnName]. Where TableName is the database table name, ColumnName is the field name, and RenamedColumnName is the renamed field name.

This parameter is optional.

INPUT pcUpdatableColumns AS CHARACTER

Comma separated list of logical values with special cases such as YES, NO, or a combination of YES,NO,YES,NO. If the value is:

- **YES** — All the columns are updatable.
- **NO** — All the columns are not updatable.
- **YES,NO,YES,YES** — The individual columns have the specified value.

Returns: LOGICAL

Notes: None.

deleteRecordStatic

Deletes a record from the specified table in the query.

Location: query.i

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 fails if there is delete validation defined on the target table. The workaround is to use a static "DELETE {table}." statement instead.

fetchFirst

Procedure that repositions the database query to the first row.

Location: query.p

Parameters: None

Notes: Requests rows to be transferred from the database query if the RowObject query is empty.

fetchLast

Procedure that repositions the Query to the last row.

Location: query.p

Parameters: None

Notes: None

fetchNext

Procedure that repositions the Query to the next row.

Location: `query.p`

Parameters: None

Notes: None

fetchPrev

Procedure that repositions the Query to the previous row.

Location: `query.p`

Parameters: None

Notes: None

fetchFirstBatch

Procedure that reads the first batch of the data object.

Location: `query.p`

Parameters: None

Notes: None

fetchNextBatch

Procedure that reads the next batch of the data object but does **not** fill the batch at the end.

Location: `query.p`

Parameters: None

Notes: None

fetchPrevBatch

Procedure that reads the next batch of the data object and fills the batch at the beginning.

Location: `query.p`

Parameters: None

Notes: None

fetchLastBatch

Procedure that reads the last batch of the data object.

Location: `query.p`

Parameters: None

Notes: None

fetchCurrentBatch

Reads the current batch of the data object.

Location: `query.p`

Parameters: None

Notes: None

filterContainerHandler

Procedure that adds a Filter link between a Filter container handler and a Filter container. `filterContainerHandler` is called from `startFilter` after the Filter container is constructed.

Location: `query.p`

Parameters:

`phFilterContainer AS HANDLE`

Handle of the Filter container.

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.

firstBufferName

Returns the first buffer reference in a where-clause expression.

Location: `query.p`

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

Returns the ROWID (converted to a character string) of the first database query row satisfying the passed query prepare string.

Location: `query.p`

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`.

indexInformation

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.

Location: query.p

Parameters:

INPUT pcQuery AS CHARACTER

INPUT plUseTableSep AS LOGICAL

INPUT pcIndexInfo AS CHARACTER

Returns: CHARACTER

Notes: None

initProps

Procedure that, during initialization, sets all the object's properties that are related to the Query.

Location: query.i

Parameters: None

Notes: None

initializeObject

Procedure that initializes code for query objects.

Location: query.p

Parameters: None

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 opens the query.

insertExpression

Inserts an expression into a buffer's WHERE clause.

Location: `query.p`

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`.

newQueryString

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 is always embedded in parentheses.

Location: `query.p`

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

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).

Location: `query.p`

Parameters:

INPUT `pcQueryString` AS CHARACTER

Complete query string to which the expression is going to be added.

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 as a result is 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

Returns a new query.

Location: query.p

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.

newWhereClause

Inserts a new expression to a specified buffer's WHERE clause.

Location: query.p

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 is corrected if it exists in the object's query. Otherwise, it needs to match.

openQuery

Function used to open the database query either for a SmartDataObject(SDO) or for another query object.

Location: `query.p`

Parameters: None

Returns: LOGICAL

Notes: The QueryString property that is modified by the addQueryWhere, assignQuerySelection and setQuerySort methods is used (if it is not blank) in the QUERY-PREPARE method before opening the query.

prepareQuery

Internal function. Prepares the database query for some querying object such as a SmartDataObject.

Location: `query.p`

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` checks and uses as input to this method.
- `setQueryWhere` calls this method and blank properties used by 1 in order to make `openQuery` NOT call this.

removeForeignKey

Removes the ForeignKey from the query string.

Location: `query.p`

Parameters: None

Returns: LOGICAL

Notes: The ForeignKey consists of ForeignKeys and ForeignValues.

removeQuerySelection

Removes query selection criteria from the QueryString property that were added by assignQuerySelection.

Location: `query.p`

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 prepares 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.

resolveBuffer

Resolves the correct qualified buffer name of the passed buffer reference.

Location: `query.p`

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.

rowidWhere

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 is returned and the expression in `pcWhere` can only reference that table.

Location: `query.p`

Parameters:

INPUT `pcWhere` AS CHARACTER

The where clause to apply to the database query to fetch the first record whose ROWID is to 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

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.

Location: `query.p`

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

startFilter

A procedure that views, starts, or both views and starts, the linked filter-source.

Location: `query.p`

Parameters: None

Notes: None

transferDBRow

A procedure that buffer-copies the database records for the current query row into the RowObject temp-table.

Location: `query.p`

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.

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.

whereClauseBuffer

Returns the buffer name of a WHERE clause expression. This function avoids problems with leading or double blanks in WHERE clauses.

Location: `query.p`

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

PRIVATE, used internally in `query.p` only.

Methods for data query objects

The following section lists and describes methods for data query objects.

addRow

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.

Location: data.p

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

askQuestion

Procedure that prompts for the maximum number of records to export and confirms the transfer of data in these records.

Location: exportdata.p

Parameters:

INPUT pcMessage AS CHARACTER

Message to display.

INPUT pcTitle AS CHARACTER

Title of message box.

INPUT plFieldChoice AS LOGICAL

Available options for DisplayFields. You can select:

- **Yes** — Displays three buttons: Displayed, All, and Cancel.
- **No** — Displays two buttons: OK and Cancel.

INPUT-OUTPUT piMaxRecords AS INTEGER

Maximum number of records to export.

OUTPUT pDisplayed AS LOGICAL

Indicates which button the user chose in the dialog box. The available outputs are:

- **Yes** — All fields.
- **No** — Displayed fields only.
- **?** — Cancel.

Notes: All text is defined as variables in the definition section.

batchRowAvailable

Checks to determine whether there are more records in the batch.

Location: data.p

Parameters:

pcMode AS CHRAR

Specifies the direction in which to look for records. Values are:

- Next
- Prev
- Current

Notes: Use this function before navigating with the normal fetch navigation methods to avoid an implicit read of a new batch in cases when you want to just navigate through one batch of data.

beginTransactionValidate

Wrapper procedure called automatically on entry into a transaction block.

Location: data.p

Parameters: None

Notes: None

cancelRow

Cancels an Add, Copy, or Save operation.

Location: data.p

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.

canNavigate

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

Location: data.p

Parameters: None

Returns: LOGICAL

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.

clientSendRows

Calls across the AppServer boundary to serverSendRows to fetch a batch of RowObject temp-table records. When the SmartDataObject (SDO) is divided between the client and AppServer, this client-side procedure runs from the generic sendRows procedure and then runs a server-side version that returns the RowObject temp-table records.

Location: data.p

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 from what is indicated by piStartRow or piRowIdent.

INPUT piRowsToReturn AS INTEGER

The number of rows in a batch.

OUTPUT piRowsReturned AS INTEGER

The actual number of rows returned. This number is either the same as piRowsToReturn or less when there are not enough records to fill up the batch.

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.

closeQuery

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.

Location: data.p

Parameters: None

Returns: LOGICAL

Notes: None

colStringValue

Returns a delimited list of values for the requested columns of the current row of the RowObject.

Location: dataextcols.p

Parameters:

INPUT pcColumnList AS CHARACTER

Comma delimited list of RowObject column names.

INPUT pcFormatOption

Format of returned values. Possible values are:

- **Blank or ?** — No formatting, buffer values only.
- **Formatted** — Formatted according to the columnFormat with right justified numeric values.
- **Trim Numeric** — Formatted according to the columnFormat with Left justified numeric values.

INPUT pcDelimiter

Optional delimiter (default CHR(1))

Notes: This function is different from colValues in that it does NOT return rowidents as the first entry and does not look in the dataSource.

colValues

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.

Location: dataextcols.p.

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.

Commit

Client-side part of the Commit function. Copies changed records into an update temp-table and sends it to the serverCommit procedure.

Location: data.p

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.

commitData

Procedure that calls undoTransaction to clean up the temp tables after the commit operation finishes.

Location: data.p

Parameters:

OUTPUT pcError AS CHAR

commitTransaction

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.

Location: data.p

Parameters: None

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.

copyRow

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.

Location: data.p

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

createData

Creates a new record for a data object.

Location: data.p

Parameters:

pcColumnNames AS CHARACTER

Comma-separated list of column names.

pcNewValues AS CHARACTER

Current values of the updated record delimited by the [DataDelimiter](#) property.

pcError AS CHARACTER

All error messages in ADM/Dynamic format.

Notes: Output data is formatted according to the **DataReadFormat** property.

createObjects

Procedure used to define the temp-tables for a dynamic SmartDataObjects.

Location: data.p

Parameters: None.

createRow

Accepts a list of values to create a new row in a RowObject temp-table.

Location: data.p

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 is invoked from here if AutoCommit is set on.

dataAvailable

Event procedure that handles changes to dataSource data, a record position, or a change to the data or record position of this object. Changes are handled by assigning a new value to the ForeignFields property, reopening the query, resetting QueryPosition, and republishing the event.

Location: data.p

Parameters:

INPUT pcRelative AS CHARACTER

Provides information about a new or changed record. Valid entries are:

- **SAME** — Performs a RETURN only because the new record is the same as the current record.
- **RESET** — Resets the status and foreign fields, and refreshes visual objects and panels for all objects that are part of the data link. This option provides more functionality than SAME and less functionality than DIFFERENT. Use this option when you want to send notification about a change in the RowObject record without having to reopen all the dependent queries.
- **VALUE-CHANGED** — Indicates that the data target has changed position. However, ensures that the query is not reopened but published as DIFFERENT from from this point so the data targets for this object are opened.
- **DIFFERENT** — Reapply Foreign Fields values except when called from a dataTarget. In that instance, handle as VALUE-CHANGED.
- **FIRST, NEXT, PREV, LAST, REPOSITIONED** — Handle the same as DIFFERENT in this version of dataAvailable.
- **TRANSFER** — Joins the child data object to the current record in a parent data object. As a result, the call only needs to be performed in the parent data object inside of a container.

NOTE: The dataTarget passes **VALUE-CHANGED** to indicate a change that does not require the value for Foreign Fields be reapplied. As a result, the source for dataAvailable does not need to be checked, and this procedure provides support for SmartBusinessObjects (SBOs) as dataSources.

dataContainerHandle

An AppServer-aware container that handles data requests and also acts as the container of **THIS** object. The container can be a standard container or an SBO.

This function also provides the required checks of the AppServer properties for **THIS** object and returns only the handle if the current or permanent state allows it to be part of a stateless request handled by another object.

Location: data.p

Returns: Handle

deleteData

Deletes an existing record of a data object.

Location: data.p

Parameters:

pcColumnNames AS CHARACTER

Comma-separated list of column names.

pcOldValues AS CHARACTER

Current values of the updated record delimited by the [DataDelimiter](#) property. The values are used to support optimistic locking and also to identify the record.

pcError AS CHARACTER

All error messages in ADM/Dynamic format.

Notes:

- The [KeyFields](#) property identifies the columns that identify the record.
- Rowid and RowIdent are supported as a value in the ColumnNames parameter.

deleteRow

Submits a row for deletion. Returns FALSE if an error occurs.

Location: data.p

Parameters:

INPUT pcRowIdent AS CHARACTER

The RowIdent of the RowObject temp-table to delete.

Returns: LOGICAL

Notes: If auto-commit is on, the row is immediately returned to the database for deletion.

describeSchema

A procedure that returns a temp-table with a schema description in it, assembled by this routine.

Location: data.p

Parameters:

INPUT pcIndexFieldList AS CHARACTER

The list of fields of interest.

OUTPUT TABLE-HANDLE hTtSchema

The handle to the ttSchema table.

Notes: None

destroyObject

A procedure that override to get rid of the datalogicobject.

Location: data.p

Parameters: None

Notes: None

destroyServerObject

A procedure that destroys the server object and retrieves its context.

Location: data.p

Parameters: None

Notes: Called from unbindServer when stateless.

doBuildUpd

A procedure that transfers changed rows into the Update Temp-Table and returns it to the Commit function (the caller function).

Location: dataext.p

Parameters: None

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

A procedure that uses FIND to find 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.

Location: dataext.p

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 p1Reopen AS LOGICAL

TRUE if the row is new, the result of a COPY or ADD.

OUTPUT pcMessage AS CHARACTER

Error message, if any.

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.

doEmptyTempTable

A procedure that empties the RowObject temp-table when the query is reopened while a transaction is active.

Location: dataext.p

Parameters: None

Notes: This routine is only needed when a transaction is active. In other cases, use the faster EMPTY-TEMP-TABLE method.

doReturnUpd

A procedure that runs from Commit on the client side. Obtains the latest state of the records in RowObjUpd from the server for refreshing data-display objects.

Location: dataext.p

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.

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 contains the refreshed db values and passes 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

A procedure that restores deleted rows.

Location: dataext.p

Parameters: None

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

A procedure that restores the row using the saved image of the unchanged RowObjUpd record.

Location: dataext.p

Parameters: None

Notes: None

doUndoTrans

Procedure that 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.

Location: dataext.p

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.

Notes: Called by the event procedure undoTransaction. Runs on the client side.

doUndoUpdate

Procedure that supports cancelRow by copying the current RowObjUpd record back to the current RowObject record.

Location: dataext.p

Parameters: None

Notes: None

endClientDataRequest

Procedure that contains logic to retrieve data properties from the Appserver after a data request. Both queries and commit are considered as data requests.

Location: data.p

Parameters: None

Notes: The purpose of this function is to encapsulate the logic for stateless and state-aware requests in one call.

endTransactionValidate

An optional validation procedure called automatically by serverCommit after completing all updates, but before exiting the transaction block.

Location: data.p

Parameters: None

Notes: None

exportData

Procedure that export the contents of a SmartDataObject (SDO) to another program.

Location: data.p

Parameters:

INPUT phDataObject AS HANDLE

The input data object handle from which to export data.

INPUT pcType AS CHARACTER

The input type. The current options are Excel or Crystal.

INPUT pcFieldList AS CHARACTER

Either a list of input fields (Visible Fields only) or a blank for all (no table prefix).

INPUT PARAMETER plIncludeObj AS LOGICAL

Indicates whether or not the input should include object fields. The options available are **Yes** or **No**.

INPU PARAMETER plUseExisting AS LOGICAL

Indicates whether for to use the currently running program (Currently Excel only). The available options are **Yes** or **No**.

INPUT PARAMETER piMaxRecords AS INTEGER

Maximum number of records to process. The unknown value (?) signal to prompt for max records and all or disp. This ensures backwards compatibility if anyone uses the old style of asking before calling printToExcel in an SmartDataObject

Notes:

- Always excludes RowObject specific fields. For example, RowNum, RowIndent, RowMod.
- Uses tableout procedure in the SmartDataObject (SDO).

fetchBatch

Procedure that transfers another batch of rows from the database query to the RowObject temp-table query, without changing the current record position.

Location: data.p

Parameters:

INPUT p1Forwards 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.

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.

fetchFirst

Procedure that 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.

Location: data.p

Parameters: None

Notes: None

fetchLast

Procedure that repositions the RowObject query to the last row of the result set. 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.

Location: data.p

Parameters: None

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 continues to be positive integers), and all other records have smaller numbers (except for additions.)

fetchNext

Procedure that 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.

Location: data.p

Parameters: None

Notes: None

fetchPrev

Procedure that 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 is only necessary when the RebuildOnReposition property is set to TRUE.)

Location: data.p

Parameters: None

Notes: None

fetchRow

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.

Location: data.p

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

Repositions the RowObject's query to the desired row, based on the corresponding database record ROWID as specified by pcRowIdent and returns values for the rows as specified in pcViewColList. The return value is a CHR(1)-delimited list of values that are formatted according to the FORMAT expression for each column. 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.

Location: data.p

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 nonexistent rowid, the query is closed and the SDO 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.
- This method resolves the row reposition on the server. As a result, the SDO can no longer determine whether the row position is invalid until after the request has been executed. However, if the RebuildOnRepos property is set to TRUE, the Temp-table is emptied *before* the request.
- The current behavior for a FIND that does not find anything when RebuildOnRepos is TRUE is to read the current batch again.

findRow

Finds a row and repositions it using a key.

Location: data.p

INPUT pcKeyValues AS CHARACTER

Comma- or chr(1)-separated list of keyfields.

Returns: LOGICAL

Notes:

- This method resolves the row reposition on the server. As a result, the SDO can no longer determine whether the row position is invalid until after the request has been executed. However, if the RebuildOnRepos property is set to TRUE, the Temp-table is emptied *before* the request.
- The current behavior for a FIND that does not find anything when RebuildOnRepos is TRUE is to read the current batch again.

findRowWhere

Finds a row and repositions to that row.

Location: data.p

Parameters:

INPUT pcColumns AS CHARACTER

For a SmartBusinessObject (SBO), 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),

For a SmartDataObject (SDO), column names (comma separated); fieldname of a table in the query in the form of RowObject.FLDNM.

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.).
- **Comma-separated list** — list for each column and value.

Returns: LOGICAL

Notes:

- This method resolves the row reposition on the server. As a result, the SDO can no longer determine whether the row position is invalid until after the request has been executed. However, if the RebuildOnRepos property is set to TRUE, the temp-table is emptied *before* the request.
- The current behavior for a FIND that does not find anything when RebuildOnRepos is TRUE is to read the current batch again.
- The logic is in the query.p super.

firstRowIds

Returns the ROWID (converted to a character string) of the first database query row satisfying the passed query prepare string.

Location: data.p

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.

hasActiveAudit

value of hasActiveAudit.

Location: data.p

Parameters: None

Returns: LOGICAL

Notes: None

hasActiveComments

value of hasActiveComments

Location: data.p

Parameters: None

Returns: LOGICAL

Notes: None

hasForeignKeyChanged

Function that determines whether or not the dataSource Foreign fields are different from the current ForeignValues.

Location: data.p

Parameters: None

Returns: LOGICAL

Notes:

- Returns TRUE if the dataSource Foreign fields are different from the current ForeignValues. If TRUE, the query needs to be reopened.
- This function is an important part of the logic used with the dataAvailable when RESET is specified.
- Returns FALSE if the dataSource has uncommitted changes.
- Used by SmartBusinessObject (SBO) commitTransaction.

hasOneToOneTarget

Returns TRUE if this DataObject has DataTargets that are updated as part of this.

Location: data.p

Parameters: None

Returns: LOGICAL

Notes: None

initializeObject

Procedure that performs SmartDataObject-specific initialization. Specifically, starts the server-side copy of this object if the AppService (partition) is defined.

Location: data.p

Parameters: None

Notes: None

initializeLogicObject

Procedure that loads the data logic procedure as a super procedure.

Location: data.p

Parameters: None

Notes: None

initializeServerObject

Procedure that initializes and sets context in the serverObject after it has been started or restarted.

Location: data.p

Parameters: None

Notes: None

isUpdateActive

Procedure that is received as a publication from container source to check whether contained objects have unsaved or uncommitted changes, including addMode.

Location: data.p

Parameters:

INPUT-OUTPUT plActive AS LOGICAL

FALSE if the ContainerTargets should be surveyed.

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.

isUpdatePending

Procedure that publishes to data-targets to check pending updates. If no updates are known to be pending (RowObjectState is not RowUpdated, if neither NewRow nor DataModified are TRUE). Otherwise returns without action.

Location: data.p

Parameters:

INPUT-OUTPUT plUpdate AS LOGICAL

False if this routine should check for pending updates.

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.

linkState

Procedure that calls linkTargetHandler in the source and ensures that only an inactive state is republished if all its other data targets are inactive. As a result, an inactive linkState can reach the highest possible point in the class hierarchy. If the ToggleDataTargets property is set to FALSE this does not happen.

Location: data.p

Parameters:

INPUT pcState AS CHARACTER

The event is republished up the groupAssignSource and the DataSource which then runs linkstateHandler if the link can be deactivated.

Notes: This method is used to republish events when they are ignored as dataTargetEvents.

LinkStateHandler

Refreshes the data object when dataAvailable (RESET) is run to activate the object again when the passed state is active and the link is inactive.

Location: data.p

Parameters:

INPUT PARAMETER pcState AS CHARACTER

INPUT PARAMETER phObject AS HANDLE

INPUT PARAMETER pcLink AS CHARACTER

Notes: Reset ensures that no changes are made to the record so that it is the same as it was before deactivation.

newRowObject

Assigns some general RowObject fields and updates LastRowNum and FirstRowNum when a new RowObject has been created.

Location: data.p

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().

obtainContextForServer

Function that returns a list of properties and sets the required client query context properties.

Location: data.p

Returns: CHARACTER

Parameters: None.

NOTE: Called from initializeServerObject or directly when the single-hit stateless procedures are called.

openDataQuery

Opens the SmartDataObject's database query based on the current WHERE clause.

Location: data.p

Parameters:

INPUT pcPosition AS CHARACTER

First or Last

Returns: LOGICAL

Notes: Currently used by SBOs after new temp-tables have been fetched.

openQuery

Opens the DataQuery.

Location: data.p

Parameters: None

Returns: LOGICAL

Notes: None

postTransactionValidate

Optional validation procedure called automatically by serverCommit after all updates complete and the transaction block ends.

Location: data.p

Parameters: None

Notes: None

prepareErrorsForReturn

Procedure that appends the RETURN-VALUE to the list of logged errors, formats the string for return to the client. Called from serverCommit.

Location: data.p

Parameters:

INPUT pcReturnValue AS CHARACTER

INPUT pcASDivision AS CHARACTER

INPUT-OUTPUT pcMessages AS CHARACTER

Notes: PRIVATE procedure.

preTransactionValidate

Optional validation procedure called automatically by serverCommit and before entering the transaction block.

Location: data.p

Parameters: None

Notes: Use this in place of TransactionValidate (maintained for backward compatibility).

printToCrystal

Procedure that transfers the data from the SDO to Crystal.

Location: data.p

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.

Notes: Always excludes rowobject specific fields, e.g. RowNum, RowIdent, RowMod. Uses tableout procedure.

pushTableAndValidate

When running from a SmartBusinessObject, acts as a wrapper for preTransactionValidate and postTransactionValidate procedures.

Location: data.p

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

refreshRow

Procedure that retrieves the current database values for a row already in the RowObject table.

Location: data.p

Parameters: None

Notes:

- Check that the current row still exists in the database before using this procedure. Attempting to refresh a deleted row can have unpredictable results.
- PUBLISHes dataAvailable (**SAME**) to cause a SmartDataViewer or Browser to display the latest values. Since the database records are never locked on read, you can use this procedure to fetch the latest values upon a request from an application. However, this cannot guarantee that the values do not change before an update.

remoteCommit

Procedure executed on a server side SmartDataObject. This is the equivalent of serverCommit, but can be run in a non- initialized object as it has INPUT and OUTPUT parameters for context.

Location: data.p

Parameters:

INPUT-OUTPUT pcContext

INPUT context is current context from the client and OUTPUT context is the new context.

INPUT-OUTPUT TABLE RowObjUpd

The Update version of the RowObject Temp-Table.

OUTPUT cMessages

A CHR(3) delimited string of accumulated messages from the server.

OUTPUT cUndoIds

List of RowObject ROWIDs whose changes must be undone because of errors. The list uses the following form to display theRowObject ROWIDs whose changes need to be undone: RowNumCHR(3)ADM-ERROR-STRING,RowNumCHR(3)ADM-ERROR

Notes: If another user modifies the database records after the original records are read, the new database values are copied into the RowObjUpd record and returned to Commit so the user interface object can display them.

remoteSendRows

A stateless version of sendRows that does no processing but runs sendRows to pass all parameters except for the context and returns the RowObject table as an output parameter to the caller that has the new batch of records created in sendRows.

Location: data.p

Parameters:

INPUT-OUTPUT pcContext

CHR(3) separated list of propCHR(4)value pairs. INPUT is the current context and OUPUTPUT is the new context. The INPUT and OUTPUT can have different properties.

INPUT piStartRow

The RowNum value of the record to start the batch to return. Typically piStartRow is a flag with a value of ? that indicates pcRowIdent should be used instead of piStartRow.

INPUT pcRowIdent

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 p1Next

Determines whether serverSendRows should start on the next record instead of what is indicated by piStartRow or piRowIdent. If TRUE, serverSendRows should start on the next record instead of what is indicated by piStartRow or piRowIdent

INPUT piRowsToReturn

The number of rows in a batch.

OUTPUT piRowsReturned

The actual number of rows returned. This number is either the same as piRowsToReturn or less if there are not enough records to fill the batch.

OUTPUT pcMessages

Used for error messages.

Notes:

- 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.
- The object should *only* be started persistently before this is called and *not* initialized because initializeObject is run *after* the context has been set.
- The caller is responsible for destroying the object.
- For more details, see synchronizeProperties and genContext.

repositionRowObject

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.

Location: data.p

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.

resortQuery

Resorts the current query.

Location: data.p

Parameters:

INPUT pcSort AS CHARACTER

The sort expression. If blank, all sorting is removed. If unknown (?), reset to the default from the base query.

Returns: LOGICAL

Notes:

- If the query is closed, the passed-in sort expression is stored in the QueryString.
- This function sorts locally.

rowAvailable

Checks RowObject availability. Encapsulates the different query position alternatives required to check for availability.

Location: data.p

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.

rowBatchAvailable

Ensures that the navigation stays within a batch when reading a batch of data. Database records are always traversed in a forward direction to ensure that records are retrieved in a consistent order.

Location: `data.p`

Parameters: None

rowObjectValidate

The client-side validation procedure for RowObject.

Location: `dataLogic.i`

Parameters: None

Notes: None

rowValues

Retrieves a list of data from all rows in the DataObject.

Location: data.p

Parameters:

INPUT pcColumns AS CHARACTER

Comma-separated list of RowObject column names.

INPUT pcFormat AS CHARACTER

The following formatting options available for the data

- **blank or ?** — Unformatted: columnValue().
- **Formatted** — Formatted without trailing blanks.
- **TrimNumeric** — Formatted without leading spaces for numeric data (left justified).
- **NoTrim** — Formatted with leading and trailing blanks.
- **&1 &2 &n** — A free form format where the number references the column in pcColumns order and indicates that the column values should be substituted as specified instead of returned as delimiter separated values.

This allows formatting data to be mixed with the returned values. For example, F.ex: '&2 (&1)', '&2 / &1' and so on. To build a list-item-pair list, you must ensure that the delimiter is in the format; F. ex: '&2 (&1)' + ',' + '&1' where ',' is also passed as a delimiter and would return a paired list where the second item of the pair is column number one.

INPUT pcDelimiter AS CHARACTER

A single char delimiter that can be one of the following:

- **?** — chr(1) !
- **blank** — single space !

Returns: CHARACTER

Notes:

- Intended for use by SmartSelect or other nonbrowser objects that need to show all rows of the SmartDataObject (SDO).
- Should not be used with large amounts of data since all data needs to fit in the return value.
- A maximum of nine columns can be passed when a substitute format is specified.
- This function reads all data without publishing dataAvailable to its data-targets. However, if the query is browsed, dataAvailable publishes to its data-targets when reposition-to-rowid is executed to return to the current record.

saveContextAndDestroy

Procedure that saves the context of the server-side SmartDataObject in support of running in Stateless mode and then destroys it.

Location: data.p

Parameters:

OUTPUT pcContext AS CHARACTER

The context string to be returned.

Notes: saveContextAndDestroy is invoked from the client side of the SmartDataObject and executed on the server side.

sendRows

Procedure that fetches the requested number of rows from the database Query and creates corresponding records in the Row Object temp-table. The batch typically starts at the row indicated by the pcRowIdent parameter, however, it can start on the row indicated by the piStartRow parameter.

Location: data.p

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 result 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 p1Next 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.

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.

serverCommit

Server-side counterpart of the client-side Commit function, used when running in n-tier (client + AppServer) mode. This procedure 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.

Location: data.p

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.

Notes: The current versions of not-updatable records, if any, are returned in the RowObject temp-table.

serverSendRows

Server-side procedure called by its counterpart (clientSendRows) when operating in n-tier mode (client, AppServer). This procedure acts as a pass-through proxy to sendRows for assembling and retrieving a batch of records. Runs in the ASHandle if in a SmartBusinessObject.

Location: data.p

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 is always <= piRowsToReturn.

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.

startServerObject

When a SmartDataObject is split and running statelessly on an AppServer, this procedure runs on the client to start the SmartDataObject on the server.

Location: data.p

Parameters: None

Notes: This override is for error handling to show error message and returns **adm-error**.

submitCommit

Called by submitRow, this procedure commits changes to the database.

Location: data.p

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 p1Reopen

TRUE if the RowObject query is to be reopened and repositioned to the RowObject record identified by pcRowIdent.

Notes: None

submitForeignKey

Called from submitRow, this procedure defines Foreign Key values for a new row.

Location: data.p

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 ForeignFields are set and added to the pcValueList, they are also added to pcUpdColumns to let the calling code know that it is permissible to update them.

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.

- You can use this procedure to automatically assign foreign key values to 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 are added to the parameter value if they are not already there. As noted, this also modifies 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

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 is invoked from here if AutoCommit is set to on.

Location: data.p

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

synchronizeProperties

This procedure 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.

Location: data.p

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.

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

This procedure outputs requested fields of SDO in standard temp-table.

Location: data.p

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

Notes: Temp table is defined in aftsdoout.i. Fields passed in are checked with a can-do to support * for **all** or **! field_name** to exclude a field. For example, ! 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!

transferToExcel

Procedure transfers the contents of the SDO to Excel.

Location: data.p

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

Notes: Always excludes rowobject specific fields, for example, RowNum, RowIdent, RowMod. Uses tableout procedure defined in here also.

undoClientUpdate

This procedure 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.

Location: data.p

Parameters: None

Notes: This is to deal with errors before commit has been run - i.e. client side validation errors.

undoTransaction

Procedure undoes any uncommitted changes to the RowObject table when the **Undo** button is pressed in the commit panel.

Location: data.p

Parameters: None

Notes: The undoTransaction calls doUndoTrans to restore the RowObject temp-table and empty the RowObjUpd temp-table.

updateAddQueryWhere

Procedure restores a query to design-time state, re-adds the filters, then adds and saves the new WHERE clause.

Location: data.p

Parameters:

INPUT pcWhere AS CHARACTER

New WHERE clause

INPUT pcField AS CHARACTER

Field to search and replace

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 manual queries for that field are removed and default 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!

updateData

Updates an existing record.

Location: data.p

pcUpdateColumnNames AS CHARACTER

Comma-separated list of column names.

pcOldValues AS CHARACTER

Current values of the updated record delimited by the [DataDelimiter](#) property. The values are used to support optimistic locking and also to identify the record.

pcNewValues AS CHARACTER

Current values of the updated record delimited by the [DataDelimiter](#) property.

pcError AS CHARACTER

All error messages in ADM/Dynamic format.

Notes: Output data is formatted according to the **DataReadFormat** property.

updateManualForeignFields

Procedure gets foreign fields and values and setManualAssignQuerySelection so that filter does not lose information.

Location: data.p

Parameters: None

Notes: None

updateQueryPosition

Procedure 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.

Location: data.p

Parameters: None

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.

updateRow

Updates an existing row.

Location: data.p

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

Procedure passes along update-related messages (to its Navigation–Source, for example.) and adjusts the DataModified property.

Location: data.p

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 can 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.

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).

Query object properties

Query Object properties provide information about query objects and their classes. This information can include whether an object is enabled, the contents of the object and so on. You can read property values and in many instances you can change property values. To read a property value, you use a **get** function, and to change a property value, you use a **set** function.

These functions conform to the following conventions:

- **get** — Uses the form `get $\textit{propnam}$` and returns the current value of the property

NOTE: This function accepts no arguments.

- **set** — Uses the form `set $\textit{propname}$` . The set function accepts a single argument—the new value for the property—and returns TRUE or FALSE depending on whether the value change succeeds.

For more information about getting and setting property values, see [Chapter 1, “ADM2 SmartObject API Reference”](#) in this guide.

AssignList

List of updatable columns whose names have been modified in the SmartDataObject. This string takes the form: <RowObjectName>,<DBFieldName>[,...][CHR(1)...] with a comma-separated list of pairs of fields for each db table, and CHR(1) between the lists of pairs.

Data Type: CHARACTER

Notes: Read only

AutoCommit

Determines whether a Commit happens on every record update. This is FALSE by default, but if set to TRUE a Commit is automatically done after any change.

Data Type: LOGICAL

Notes: Write only

CacheDuration

The number of seconds the DataObject's data is valid on the client. The data's validity is defined in relation to its retrieval from the server, not from when it was last referenced on the client. If set to 0, the data is not cached. If set to the unknown value (?), the data is valid for a complete session or until programatically cleared or refreshed.

Data Type: INTEGER

Notes: Read and Write

CalculatedColumns

Comma-delimited list of the calculated columns for the SmartDataObject. To obtain this information, the values of DataColumnnsByTable and Tables are examined.

Data Type: CHARACTER

Notes: Read only

CheckCurrentChanged

Determines whether the DataObject code should check whether the database rows being updated have been changed since read.

Data Type: LOGICAL

Notes: Read and Write

CheckLastOnOpen

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.

Data Type: LOGICAL

Notes:

- The DataColumnsByTable 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 becomes equal to Name<CHR(1)>OrderNum,OrderDate.
- This function replaces CHR(1) with ", " returns just a comma-delimited list.

Notes: Read and Write

ClientProxyHandle

Character version of the client-side SDO handle. The string containing the client-side SDO procedure handle.

Data Type: CHARACTER

Notes: Write only

CurrentRowModified

Identifies whether or not the current RowObject row has been modified. If TRUE, the current RowObject row has been modified. If there is no current RowObject record, then CurrentRowModified returns "?".

Data Type: LOGICAL

Notes: Read only

DataColumns

Comma separated list of the columnNames for the SmartDataObject (SDO).

Data Type: CHARACTER

Notes:

- Read and Write.
- The DataColumnsByTable 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 OrderDate field from Order are selected, then the property value becomes equal to Name<CHR(1)>OrderNum,OrderDate.
- This function replaces CHR(1) with ", " and returns just a comma separated list.

DataColumnsByTables

A comma separated list of the columnNames delimited by CHR(1) to identify which columns are from which table.

Data Type: CHARACTER

Notes:

- Read and Write
- To get a comma separated list of just columnNames for a SmartObject, use DataColumn.

DataDelimiter

Delimiter for values passed to receiveData and output for the input-output in updateData and createData.

Data Type: CHARACTER

Notes: Read and Write

DataFieldDefs

Name of the include file in which the field definitions for this SmartDataObject's SDO) RowObject table are stored.

Data Type: CHARACTER

Notes: Read only

DataHandle

SmartBusinessObject version of getDataHandle is run from a browser to get the query from the contained Data object.

Data Type: HANDLE

Notes: Read only

DataLogicObject

Handle of the logic procedure that contains business logic for the data object.

Data Type: HANDLE

Notes: Read only

DataModified

Flag that indicates whether the current SCREEN-VALUES have been modified but not saved.

Data Type: LOGICAL

Notes:

- Read and Write
- 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.

DatalsFetched

Flag indicating whether data has been fetched for the SmartDataObject (SDO). The SmartBrowserObjec (SBO) t sets this to TRUE in the SDO when it has fetched data for the SDO to prevent the SDO from using another server call to fetch the data it already has.

Data Type: LOGICAL

Notes:

- Read and Write
- This is checked in query.p dataAvailable and openQuery is skipped if its TRUE. It is immediately turned off after it is checked.

DataLogicProcedure

Name of the logic procedure that contains business logic for the data object.

Data Type: CHARACTER

Notes:

- Read and Write
- The DataLogicProcedure is added as a super procedure when it is set using the setDataLogicProcedure. As a result, if other properties are set before the super procedure is added, you cannot override the property settings in the logic procedure.

DataQueryBrowsed

Identifies whether or not this SmartDataObject's query is being browsed by a SmartDataBrowser. If TRUE, this SmartDataObject's query is being browsed.

Data Type: LOGICAL

Notes: Read and Write

DataQueryString

The DataQueryString used to prepare the RowObject query.

Data Type: CHARACTER

Notes: Read and Write

DataReadBuffer

Value of DataReadBuffer.

Data Type: HANDLE

Notes: Read and Write

DataReadColumns

A comma-separated list of columns to pass to the registered DataReadHandler when traversing the query.

Data Type: CHARACTER

Notes: Read and Write

DataReadHandler

The handle of a procedure that has been registered to receive output from the object during data read.

Data Type: HANDLE

Notes: Read and Write

DataSignature

A character string that lists the integers corresponding to the datatypes of the fields in a RowObject temp-table. This string is used to compare objects for equivalence as follows:

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

Data Type: CHARACTER

Notes: Read only

DbNames

Property that contains a comma separated list of DbNames that corresponds to the tables in the query objects.

Data Type: CHARACTER

Notes: Read and Write

DestroyStateless

Determines whether the persistent SDO should be destroyed on stateless requests.

Data Type: LOGICAL

Notes:

- Read and Write
- The value of this property should not be changed during a session. If a change is necessary, the AppServer broker must be restarted or all running AppServer agents trimmed.

DisconnectAppServer

Determines whether the persistent SDO disconnects the AppServer.

Data Type: LOGICAL

Notes: Read and Write

EnabledObjFldsToDisable

Property that controls whether or not to disable nondatabase objects when the data fields are disabled. You can enter one of the following:

- **None** — Nondatabase objects remain enabled when the fields are disabled.
- **All** — Nondatabase objects are disabled in view mode
- **Comma separated list** — A comma-separated list of nondatabase object names that you want disabled in view mode.

Data Type: CHARACTER

Notes:

- Read and Write
- The property only applies to nondatabase objects that have been defined as enabled in the master. See [EnabledObjFlds](#) for more information.
- You can edit this property using the viewer's **Instance Property** dialog box.

EnabledTables

List of the database tables that have enabled fields.

Data Type: CHARACTER

Notes: Read only

FillBatchOnRepos

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 result set where an entire batch would not be retrieved.

Data Type: LOGICAL

Notes: Read and Write

FilterWindow

Name of the partition, if any, on which this object runs.

Data Type: CHARACTER

Notes: Read and Write

FirstResultRow

Pointer to the first record in the current batch. It is currently made up of the first record's RowNum and RowIdent separated by a semi-colon. Though the content of this property can change, the property **must never** be changed by application code.

Data Type: CHARACTER

Notes: Read and Write

FirstRowNum

Temp-table row number of the first row.

Data Type: INTEGER

Notes: Read and Write

FilterActive

Flag indicating whether or not the dataSource has a logical filter and whether or not a filter is active. If TRUE, a filter is active.

Data Type: LOGICAL

Notes: Read and Write

FilterAvailable

Flag indicating whether or not a filter is available.

Data Type: LOGICAL

Notes: Read and Write

ForeignFields

Sets the Foreign Fields property of the object and removes the current values for Foreign Fields from the current query if they have been applied.

Data Type: CHARACTER

Notes:

- Read and Write
- When an SBO is the DataSource of another data object (SDO or SBO), the DataTarget's ForeignFields property must be qualified with the SDO instance name of the parent SBO.

ForeignValues

Identifies the most recently received values for foreign field received by for the dataAvailable property. The values are character strings formatted according to the field format specification and are separated by the CHR(1) character.

Data Type: CHARACTER

Notes: Read only

GroupAssignSource

Handle of the object's GroupAssign source.

Data Type: HANDLE

Notes: Read and Write

GroupAssignSourceEvents

Comma-separated list of the events this object wants to subscribe to in its GroupAssign source.

Data Type: CHARACTER

Notes: Read and Write

GroupAssignTarget

Handle, in character format, of the object's GroupAssign target.

Data Type: CHARACTER

Notes: Read and Write

GroupAssignTargetEvents

Comma-separated list of the events this object wants to subscribe to in its GroupAssign target.

Data Type: CHARACTER

Notes: Read and Write

IndexInformation

Index Information formatted as the 4 GL index-information attribute, but with RowObject column names, CHR(1) as index separator, and CHR(2) as table separator.

Data Type: CHARACTER

Notes:

- Read and Write
- Intended for internal use by other index info functions, which use this as input to indexInformation(). (Unmapped columns are returned fully qualified!) 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.

InternalEntries

Internal entries of SDO as internal entries cannot be accessed for remote proxy procedures.

Data Type: CHARACTER

Notes: Read only

KeyFields

Comma-separated list of the key fields. The `indexInformation` is used to try to figure out the default `KeyFields` list, but this is currently restricted to conditions:

- The First Table in the join is the only enabled table.
- All the fields of the index are present in the SDO.
- The following index might be selected:
 - Primary index if unique
 - First Unique index
- There is currently no check whether the field is mandatory.

Data Type: CHARACTER

Notes: Read and Write

LastCommitErrorKeys

Property that provides information about records that failed during the last data commit. For:

- **SDOs** —A comma-delimited list of the key values of the records that failed to be committed. The `KeyFields` property of the SDO holds the key field names.
- **SBOs** — A semicolon-delimited list of the values of each individual contained SDO that has failed records.

Data Type: CHARACTER

Notes:

- Read and Write
- A blank indicates that the last commit was successful.

LastCommitErrorType

Property that identifies the type of error encountered the last time data was committed:

- **Blank** — The last commit was successful.
- **Unknown** — A commit was not attempted after run.

- **Conflict** — A locking conflict occurred.
- **Error** — An unspecified error occurred.

Data Type: CHARACTER

Notes:

- Read and Write
- Currently used to identify a Conflict error when using the UpdateData procedure. See [Update Data](#) for additional information.

LastDbRowIdent

Identifies the database rowid(s) for the last row fetched row. This value is unknown if the last row has not been fetched.

Data Type: CHARACTER

Notes: Write only

LastCommitErrorType

Property that identifies the type of error encountered the last time data was committed.

- **Blank** — The last commit was successful.
- **Unknown** — A commit was not attempted after run.
- **Conflict** — A locking conflict occurred.
- **Error** — An unspecified error occurred.

Data Type: CHARACTER

Notes:

- Read and Write
- Currently used to identify a Conflict error when using the UpdateData procedure. See [“updateData”](#) for additional information.

LastResultRow

Pointer to the last record in the current batch. It is currently made up of the last record's RowNum and RowIdent separated by a semi-colon. Though the content of this property can change, the property **must never** be changed by application code.

Data Type: CHARACTER

Notes: Read and Write

LastRowNum

Temp-table row number of the last row. This value is unknown if the last row has not been fetched.

Data Type: INTEGER

Notes: Read and Write

LogicBuffer

Handle of the data-logic table, if possible.

Data Type: HANDLE

Notes: Read and Write

ManualAddQueryWhere

Manual calls to addQueryWhere. This value is reapplied by the filter when needed to ensure the original query remains intact.

Data Type: CHARACTER

Notes:

- Read and Write
- Value is pcwhere + CHR(3) + pcbuffer or empty or "?" + CHR(3) + pcandor
- Multiple entries are supported, separated by CHR(4).

ManualAssignQuerySelection

Manual calls to assignQuerySelection. This value is reapplied by the filter when needed to preserve the original query.

Data Type: CHARACTER

Notes:

- Read and Write
- Value is pccolumns + CHR(3) + pcvalues + CHR(3) + pcoperators.
- Multiple entries are supported, separated by chr(4).

ManualSetQuerySort

Manual calls to setQuerySort. This value is reapplied by the filter when needed to preserve the original query.

Data Type: CHARACTER

Notes: Read and Write

NavigationSource

Handle of the query object's navigation source.

Data Type: CHARACTER

Notes: Read and Write

NavigationSourceEvents

Comma-separated list of the events this object wants to subscribe to in its NavigationSource.

Data Type: CHARACTER

Notes: Read only

NewRow

Indicates whether the current RowObject record is a new record or a copy of an existing record. If the value for this property is:

- **TRUE** — Then either a new record or a copy of an existing record was added to the database.
- **?** — Then there is no current RowObject.

This (SmartBusinessObject) SBO property is similar to DataHandle and is run from a browser to get the query from the contained Data object.

For more information, see [DataHandle](#).

Data Type: LOGICAL

Notes: Read only

OpenOnInit

Flag indicating whether or not the object's database query should be opened automatically when the object is initialized. The default value is yes.

Data Type: LOGICAL

Notes: Read and Write

OpenQuery

Original design WHERE clause for a database query. The value for this property is used by QueryWhere to manipulate data.

Data Type: CHARACTER

Notes:

- Read and Write
- OpenQuery is called from the client on AppServer.
- No matter how the query is dynamically modified, it can be reset to its original state using the value of this property in a QUERY-PREPARE method.
- For more information, see [RebuildOnRepos](#).

PhysicalTable

Property that contains the physical names that corresponds to the tables used to build the base query.

Data Type: CHARACTER

Notes:

- Read and Write
- The names used in the query are defined in the corresponding Tables property.
- If the query is defined with dbname, it must be qualified with database name.

PropertyList

List of properties taken from a CHR(3)-delimited list of "propCHR(4) value" pairs. Used internally to support running in Stateless mode.

Data Type: CHARACTER

Notes: Write only

QueryContainer

Indicates whether or not the Container is a QueryObject. Use this information to determine if an SBO handles the transaction.

Data Type: CHARACTER

Notes: Read only

QueryHandle

Handle of the database query of the database object.

Data Type: HANDLE

Notes: Read only

QueryOpen

Returns TRUE if the Query is currently open.

Data Type: LOGICAL

Notes:

- Read only
- On the server side, it is the database query of the query object.
- On the client side, it is the RowObject query.

QueryPosition

Identifies the state and position of the current record in the data object and is actively used by other objects like the Navigation toolbar. Valid values are: **FirstRecord**, **LastRecord**, **NotFirstOrLast**, **NoRecordAvailable**, and **NoRecordAvailableExt**.

Data Type: CHARACTER

Notes: Read and Write

QueryRowIden

The row indentation to be used to position an SDO query when it is first opened.

Data Type: CHARACTER

Notes:

- Read only
- Generally used to save the position of a query when it is closed so that position can be restored on reopen.

QuerySort

Property that contains the sorting criteria (BY phrase) of the database query. The value of this property is then stored in the QueryString property.

Data Type: CHARACTER

Notes:

- Read and Write
- If you need to set the query sort criteria using a field that was renamed in a SmartDataObject (SDO), you must qualify the field name using RowObject. For example, if a SmartDataObject for the customer table includes the field **customer.contact** and that field was renamed to **custContact** in the SDO, you can pass the **custContact** field to setQuerySort either as the field **RowObject.custContact** or the database column name **contact**.
- The sort expression is removed if blank is passed as input.
- setQueryWhere overrides the entire query string including any previous setting of the QuerySort.

QueryString

Pending query string to use in the next openQuery. This value is updated on the client by query manipulation functions.

Data Type: CHARACTER

Notes:

- Read and Write
- This property should not be changed by code but is useful for debugging code.
- 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 calls prepareQuery with this property.

QueryWhere

WHERE clause that is appended to the basic static database query definition on either the client or server side, or a completely modified Where clause.

Data Type: CHARACTER

Notes: Read and Write

RebuildOnRepos

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 result set.

Data Type: LOGICAL

Notes: Read and Write

RowIdent

Comma-delimited character string containing the ROWIDs of the database records that are the source of the RowObject record.

Data Type: CHARACTER

Notes: Read only

RowObjectState

Indicates whether there are uncommitted updates in the object. Valid return values are **NoUpdates** and **RowUpdated**.

Data Type: CHARACTER

Notes: Read and Write

RowObjectTable

Temp-table handle of the RowObject table.

Data Type: HANDLE

Notes: Read and Write

RowObjUpd

Handle of the temp-table buffer where updates are stored.

Data Type: HANDLE

Notes: Read only

RowObjUpdTable

Handle of the RowObjUpd temp-table.

Data Type: HANDLE

Notes: Read only

RowsToBatch

Number of rows to be transferred from the database query into the RowObject temp-table in a single operation.

Data Type: INTEGER

Notes: Read and Write

SchemaLocation

Determines where an DynSDO gets its field defintions. The possible values are:

- **ENT** — Use the Repository's Entity Cache. (Default)
- **DLP** — Copy field definitions from the Data Logic Procedure buffer.
- **BUF** — Copy field definitions from the database buffer.

Data Type: CHARACTER

Notes:

- Read and Write
- Before using this property, read the *Progress Dynamics 2.1A Performance* whitepaper.

ServerSubmitValidation

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 executes SubmitValidation itself.

Data Type: LOGICAL

Notes: Read and Write

ShareData

Setting this to TRUE means that cached data can be shared.

Data Type: LOGICAL

Notes: Read and Write

Tables

Property that contains a comma separated list of tables from which data is retrieved. The data is retrieved either from a SmartDataObject or a database.

Data Type: CHARACTER

Notes:

- Read and Write
- Qualified with database name if the query is defined with dbname.
- Currently this property is a design-time property whereas before it was resolved from the actual query.
- 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.
- The names in the list references to the tables as defined in the data object and referenced in the query and each entry must be unique. Therefore, the names might be buffer names that differ from the actual physical names, and the physical name is defined in the corresponding PhysicalTables property.

ToggleDataTargets

Property used to determine whether or not dataTargets should be switched between **on** and **off** in LinkState.

Data Type: Logical

Notes:

- Read and Write
- If set to FALSE, dataTargets should not be switched between **on** and **off** based on the **active** or **inactive** parameter in LinkState.

UseDBQualifier

Indicates whether table references are qualified with database. If TRUE, table references are qualified with database.

Data Type: LOGICAL

Notes: Read only

UpdateFromSource

Identifies whether or not an object should be updated as one-to-one of the datasource updates. TRUE if this object should be updated.

Data Type: LOGICAL

Notes: Read and Write

UpdatableColumns

A comma separated list of the updatable columns for this SmartDataObject (SDO), or a list of all updatable columns of contained data objects qualified by their object names.

Data Type: CHARACTER

Notes:

- Read only
- If you want a list of updatable columns in different tables delimited by CHR(1), instead of by a comma, that identifies which columns are from which table in the event of a join, use the UpdatableColumnsByTable property.

UpdatableColumnsByTable

A list of updatable columns in different tables, delimited by CHR(1) instead of a comma, that identifies 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 updatable, then this property value equals Name<CHR(1)>OrderNum,OrderDate.

Data Type: CHARACTER

Notes:

- Read and Write.
- If you want a comma separated list of the updatable columns for the SmartDataObject (SDO), use the UpdatableColumns property.

WordIndexedFields

Comma-separated list of RowObject fields that are mapped to database fields that have a word indexed.

Data Type: CHARACTER

Notes: Read only

Column properties for query objects

There are a number of column properties available for which you can obtain and set (write) field values. All of these properties can be read and some of them can be set. To read and set column properties, you use the following prefixes with the specific column property:

- **Column** — Use to read the value of a specific column property. For example, if you want to read the value of the ColumnLabel property, you would specify **ColumnColumnLabel**. This would return the label of the column you specify. To obtain the data type for a specific column property, you would specify **ColumnDataType**. This would return the data type of the specified column.
- **Assign** — Use to set the value of a specific column property. For example, if you want to set the label value for a specific column, you specify **assignColumnColumnLabel**. This would set the label value for the column you specify. To set a value for the column format, you would specify **assignColumnFormat**.

For additional information, see the *Progress Dynamics Programming Handbook*.

Table 5–1 lists the column properties for query objects, provides a brief description of each property, indicates whether the property can be read or set (write), and lists the data type for each property.

Table 5–1: Column properties for query objects

(1 of 2)

Column property	Description	Read	Write	Data type
ColumnLabel	Label of the specified column.	Yes	Yes	CHARACTER
DataType	Data type of the specified column.	Yes	No	CHARACTER
DbColumn	Qualified database name (DatabaseName.TableName.FieldName) mapped to the RowObject column identified by pcColumn.	Yes	No	CHARACTER
Extent	Extent of the specified RowObject column.	Yes	No	INTEGER
Format	Format of the specified RowObject column.	Yes	Yes	CHARACTER
Handle	Handle of the specified RowObject column.	Yes	No	HANDLE
Help	Help text for the specified RowObject column.	Yes	Yes	CHARACTER
Initial	Initial value for a specified field as a character string with the field FORMAT applied.	Yes	No	CHARACTER
Label	Label for the specified column.	Yes	No	CHARACTER
Modified	Indicates whether or not a specified RowObject field has been modified. TRUE if modified, FALSE if not modified.	Yes	No	LOGICAL

Table 5–1: Column properties for query objects

(2 of 2)

Column property	Description	Read	Write	Data type
PrivateData	Private Data property of the specified column.	Yes	Yes	CHARACTER
QuerySelection	CHR(1)-separated string with all operators and values, but no field names, that have been added to the Query for this column.	Yes	No	CHARACTER
ReadOnly	Identifies whether or not the specified column is in the list of columns that you can update. TRUE, if the column is in the list and FALSE, if not in the list.	Yes	No	LOGICAL
StringValue	String Value of the specified column in the SmartDataObject RowObject buffer.	Yes	No	CHARACTER
Table	Database table name of the specified column.	Yes	No	CHARACTER
ValExp	Validation expression for the specified column in a RowObject temp-table.	Yes	Yes	CHARACTER
ValMsg	Validation message text for the specified column of the SmartDataObjects RowObject temp-table.	Yes	Yes	CHARACTER
Value	Raw (unformatted) character value of the specified field in a RowObject temp-table.	Yes	No	CHARACTER
Width	Width in character units of the specified column of the SmartDataObjects RowObject temp-table.	Yes	No	DECIMAL

Toolbar Objects and Their Methods and Properties

This chapter lists and describes the methods (internal procedures and functions) and properties used for Toolbar Objects. Refer to [Figure 1-1](#) to see the inheritance hierarchy for each object class.

NOTE: For information specific to the WebSpeed environment (see the [“Alphabetical Listing Of WebSpeed-specific API Routines”](#) chapter).

Methods for toolbar object actions

This section lists and describes the methods for Toolbar Object actions.

canFindAction

Returns TRUE if an action exists.

Location: toolbar.p

Parameters:

INPUT pcAction AS CHARACTER

 Name of the action.

Returns: LOGICAL

Notes: None

canFindCategory

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

Location: toolbar.ptoolbar.p

Parameters:

INPUT pcCategory AS CHARACTER

 The category (action) of interest.

Returns: LOGICAL

Notes: None

categoryLink

Returns the value of Link for the specified category.

Location: toolbar.p

Parameters:

INPUT pcCategory AS CHARACTER

 The category of interest.

Returns: CHARACTER

Notes: None

checkRule

Checks the rules of an action against the target.

Location: toolbar.p

Parameters:

INPUT pcRule AS CHARACTER

The rules for the target.

INPUT phHandle AS HANDLE

Handle of the dynamic Buffer with the rules or the handle of the target.

INPUT plDefault AS LOG

Default is used when the function or property is not found, or when the function returns ?.

Returns: CHARACTER

Notes: This functionality is duplicated in panel.p

defineAction

Creates a temp-table entry defining an action, possibly overwriting the existing definition.

Location: toolbar.p

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.

displayActions

Utility procedure that displays a dialog showing all the Actions currently defined.

Location: toolbar.p

Parameters: None

Notes: Can be executed by selecting displayActions from the PRO*TOOLS procedure object viewer for the desired SmartContainer.

initAction

Procedure that defines all default actions for the SmartToolbar.

Location: toolbar.p

Parameters: None

Notes: Called once from initializeObject

initializeObject

Procedure that calls initAction to define the set of actions used by SmartToolbars.

Location: toolbar.p

Parameters: None

Notes: None

Panel methods for toolbar objects

This section lists and describes the methods for panel Toolbar Objects.

activeTarget

Return the targets of the specified link type.

Location: panel.p

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.

countButtons

Procedure that walks the widget tree of the frame, counts all the buttons, and sets the ButtonCount property.

Location: panel.p

Parameters: None

Notes: This sets the Panel property ButtonCount. For internal use only.

disableActions

Disables a specified list of actions.

Location: panel.p

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.

enableActions

Enables a specified list of actions.

Location: panel.p

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.

enableObject

Procedure that restores the panel buttons to their previous state whenever the panel is re-enabled.

Location: panel.p

Parameters: None

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.

hasActiveGATarget

Checks to determine whether any group assign targets are active.

Location: panel.p

Parameters:

INPUT phObject AS HANDLE

phObject – Procedure object that is tableioTarget and potential GroupAssignTarget.

Returns: LOGICAL PRIVATE

Notes: None

initializeObject

Procedure used for SmartPanel-specific initialization.

Location: panel.p

Parameters: None

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**.

linkState

Receives messages when an object linked to this object becomes active or activeTarget (when it is viewed) or inactive or inactiveTarget (when it is hidden).

Location: panel.p

INPUT PARAMETER pcState AS CHARACTER

State of a linked object. Valid states are:

- active
- activeTarger
- inactive
- inactiveTarget

Notes: None

loadPanel

Procedure that loads actions.

Location: panel.p

Parameters: None

Notes: None

onChoose

Procedure that reads actions and parameters from Action class (Repository) and publishes the call built from them.

Location: panel.p

Parameters:

INPUT pcAction AS CHARACTER

The action to be undertaken.

Notes: None

queryPosition

Procedure that 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.

Location: panel.p

Parameters:

INPUT pcState AS CHARACTER

Panel State.

Notes: For an Update panel, first/last/notfirstorlast/only are all forms of recordAvailable. This event is ignored if the source of the event is hidden.

resetCommit

Procedure that resets the commit target actions.

Location: panel.p

Parameters: None

Notes: None

resetNavigation

Procedure that resets the navigation target actions.

Location: panel.p

Parameters: None

Notes: None

resetTableio

Procedure that encapsulates ALL tableio settings.

Location: panel.p

Parameters: None

Notes: Can probably replace ALL other events except:

- linkChanged('inactive').
- updateState(updateComplete) and Tableiomode = update backfire of updateMode()

resetTargetActions

Procedure that resets action sensitivity and visibility.

Location: panel.p

Parameters:

INPUT pcLink AS CHARACTER

The link type whose actions are to be reset.

Notes: Overridden in toolbar.p with use of temp-tables and additional logic for alternate image.

resizeObject

Procedure that changes the size and shape of the panel. This routine spaces the buttons to fill the available space.

Location: panel.p

Parameters:

INPUT pd_height AS DECIMAL

The desired height (in rows).

INPUT pd_width AS DECIMAL

The desired width (in columns).

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.

sensitizeActions

Turns sensitivity on or off for a specified list of actions.

Location: panel.p

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

targetActions

Return the actions of a target.

Location: panel.p

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

updateState

Procedure that receives state-message events related to record updates.

Location: panel.p

Parameters:

INPUT pcState AS CHARACTER

 Update state can be **Update** or **UpdateComplete**

Notes:

- For **Update**, a Navigation Panel set its PanelState property to disable--all, to prevent navigation during an update operation. A Save or Update panel sets 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 restores the prior PanelState (having been disabled when the update began). A Save or Update panel sets its state to **add--only** if no rows are available in the current result set, to **disable--all** if no rows are available and there is also no available parent row for this result set (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.

viewHideActions

Procedure that views and hides static actions.

Location: panel.p

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

Notes: Dynamic version in toolbar.p.

Toolbar methods for toolbar objects

This section list and describes the methods for Toolbar objects.

actionCanRun

Determines whether or not a target is valid and a procedure exists. If TRUE, the target is valid and the procedure exists.

Location: toolbar.p

Parameters:

INPUT pcAction AS CHARACTER

The action of interest, of type **RUN**.

Returns: LOGICAL

Notes: Called from buildMenu and createToolbar.

actionCaption

Override action class and caption.

Location: toolbar.p

Parameters:

INPUT pcAction AS CHARACTER

The replacement action.

Returns: CHARACTER

Notes: None

actionCategoryIsHidden

Determines whether or not the category for an action is hidden. If TRUE, the category is hidden.

Location: toolbar.p

Parameters:

INPUT pcAction AS CHARACTER

The action to be evaluated.

Returns: LOGICAL

Notes: None

actionChecked

Determines whether or not a call to *getpropname* returns a value that matches the actions checked. If TRUE, the call to *getpropname* returns a value that matches the actions checked.

Location: toolbar.p

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.

actionLabel

Overrides the label of an action.

Location: toolbar.p

Parameters:

INPUT pcAction AS CHARACTER

The overriding action.

Returns: CHARACTER

Notes: None

actionPublishCreate

Subscribes create events to objects.

Location: toolbar.p

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.

actionTarget

Handle of the target. By default, this is the handle of CONTAINER-SOURCE.

Location: toolbar.p

Parameters:

INPUT pcAction AS CHARACTER

The action of interest.

Returns: HANDLE

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

actionTooltip

Overrides the tooltip of an action.

Location: toolbar.p

Parameters:

INPUT pcAction AS CHARACTER

The overriding action.

Returns: CHARACTER

Notes: None

buildAllMenus

Procedure that builds all submenu branches before the persistent trigger Menu-Drop creates them at the mouse click. As a result, accelerators (keyboard shortcuts) are enabled from startup.

Location: toolbar.p

Parameters: None

Notes: None

categoryActions

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

Location: toolbar.p

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.

constructMenu

Runs from initializeMenu when toolbar instance is first instantiated. Returns TRUE if successful.

Location: toolbar.p

Parameters: None

Returns: LOGICAL

Notes: This procedure must only be run ONCE and must be run before buildToolbar.

constructToolbar

Construct the toolbar from Repository data.

Location: toolbar.p

Parameters: None

Returns: LOGICAL

Notes: This realizes bands read from the repository. Non-repository toolbars are created with createToolbar. Both use createToolbarAction to realize the actions/widgets.

createMenuTable

Creates a temp-table for a menu.

Location: toolbar.p

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

Create the temp-table for a menu.

Location: toolbar.p

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, someMenu record might exist with "*" as parent.

createToolbar

Create a toolbar from non-Repository data.

Location: toolbar.p

Parameters:

INPUT pcActions AS CHARACTER

Comma-separated list of actions or action-groups to include. **RULE** means a separator line.

Returns: LOGICAL

Notes: None

filterState

Procedure published from NAVIGATION-TARGET that enables the panel to filter action when it is linked.

Location: toolbar.p

Parameters:

INPUT pcState AS CHARACTER

Valid value: **FilterAvailable**.

hideObject

Procedure that hides the object.

Location: toolbar.p

Parameters: None

Notes: For sizing reasons, the label is not really a part of the object, but added as a text widget to the parent frame.

imageName

Returns the name of the specified button image. This is published through the container link and used mainly to validate that a FALSE value received in updateActive can be used to set 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.

Location: toolbar.p

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 NOT** intended to be called directly, but part of the logic that updates UpdateActive.

initializeObject

Procedure that initializes the toolbar by creating all dynamic buttons and menus.

Location: toolbar.p

Parameters: None

Notes: None

insertMenu

Creates a set of menu items or submenus under a parent menu.

Location: toolbar.p

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 p1Expand 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

linkRuleBuffer

Creates the dynamic table used to check the rules against a target.

Location: `toolbar.p`

Parameters:

`pcLink AS CHAR`

 Name of the link.

`phTarget AS Handle`

 Handle of any Target.

Returns: `HANDLE`

Notes:

- The handle for the target is not created again if the link is changed. Instead, the same handle is used based on the assumption that all objects of the same type have the same API. If this is not TRUE, and the handle is not found, the data type is set to CHARACTER.
- This functionality is duplicated in `panel.p`

linkState

Receives messages when an object linked to this object becomes active or activeTarget (when it is viewed) or inactive or inactiveTarget (when it is hidden).

Location: `toolbar.p`

Parameters:

`INPUT PARAMETER pcState AS CHARACTER`

 State of a linked object. Valid states are:

- `active`
- `activeTarget`
- `inactive`
- `inactiveTarget`

Notes: Resets panel buttons to reflect the state of the object.

modifyDisabledActions

Modifies the DisabledActions property and makes it possible to permanently disable actions independent of state changes.

Location: toolbar.p

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 are immediately disabled and subsequent calls to enableActions cannot enable them again. If pcMode = **REMOVE**, the actions that are removed from the list are enabled the next time they are called with enableActions.

onChoose

Procedure that is a persistent trigger code for dynamic menu and toolbar objects.

Location: toolbar.p

Parameters:

INPUT pcName AS CHARACTER

The Action identifier.

Notes: None

onMenuDrop

Procedure used to identify what logic to execute when a submenu is dropped.

Location: toolbar.p

Parameters:

INPUT pcAction AS CHARACTER

The action's unique identifier.

Notes: Added as a persistent trigger when the submenu is created.

onValueChanged

Procedure that is a persistent trigger for toggle menu-items.

Location: toolbar.p

Parameters:

INPUT pcAction AS CHARACTER

Name of action.

Notes: Added as a persistent trigger when the submenu is created. Currently supports only logical properties.

queryPosition

Procedure that 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.

Location: toolbar.p

Parameters:

INPUT pcState AS CHARACTER

State of the Panel.

Notes: For an UPDATE panel, first/last/notfirstorlast/only are all forms of recordAvailable.

repositionObject

Procedure that 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.

Location: toolbar.p

Parameters:

INPUT pdRow AS DECIMA

INPUT pdCol AS DECIMAL

Notes: The file toolbar.i defines EXCLUDE-repositionObject (for smart.i).

resetTargetActions

Procedure that resets target actions of some kind for an object.

Location: toolbar.p

Parameters:

INPUT pcLink AS CHARACTER

The kind of actions to reset.

Notes: None

resetToolbar

Procedure that resets toolbar actions by calling resetTargetActions.

Location: toolbar.p

Parameters: None

Notes: None

resizeObject

Procedure that overrides the size after a resize. The only allowed sizing is shrinking to the last button.

Location: toolbar.p

Parameters:

INPUT pd_height AS DECIMAL

INPUT pd_width AS DECIMAL

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.

retrieveBandsAndActions

Encapsulates the retrieval and population of the toolbar temp-tables.

Location: toolbar.p

Parameters:

INPUT pcToolbarList AS CHARACTER

A comma delimited list of toolbars for which to retrieve menu data. If a blank or unknown value is passed in, then no toolbars are retrieved.

INPUT pcObjectList AS CHARACTER

A comma delimited list of objects for which to retrieve menu data. If a blank or unknown value is passed in, then no objects are retrieved.

INPUT pcBandList AS CHARACTER

A comma delimited list of bands for which to retrieve data. If a blank or unknown value is passed in, then no bands are retrieved.

Returns: LOGICAL

Notes: This function only retrieves data. It does not ensure uniqueness of the data. There are normalize functions to ensure that the data is unique.

rowObjectState

Procedure that is 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.

Location: toolbar.p

Parameters:

INPUT pcState AS CHARACTER

Valid values: **NoUpdates**, **RowUpdated**.

Notes: This could be a property, but at present the solution is to just check the state of the Commit button.

runInfo

Procedure that returns the necessary information for RUN or PROPERTY.

Location: toolbar.p

Parameters:

INPUT pcAction AS CHARACTER

Action Id where type = RUN.

OUTPUT pocProcedure AS CHARACTER

Target Procedure.

OUTPUT pocParam AS CHARACTER

Notes: Encapsulates the logic that looks in the parent for this information, if it is not defined in the action itself.

storePendingSensitivity

Procedure used to store pending sensitivity settings used to resolve timing issues where sensitivity is set before the toolbar is built.

Location: toolbar.p

INPUT pcActions AS CHARACTER

INPUT plSensitive AS LOGICAL

Notes: None

targetActions

Returns the actions that apply to a specified target.

Location: toolbar.p

Parameters:

INPUT pcLinkType AS CHARACTER

The type of target

Returns: CHARACTER

Notes: None

updateActive

Procedure that is published from Toolbar-Target (the container) to signal that some of its objects have changed state. Runs resetToolbar in TARGET-PROCEDURE.

Location: toolbar.p

Parameters: None

Notes: None

updateCategoryLists

Procedure that updates the lists of available categories for the toolbar. Called from loadBands at design time. The result is stored in AvailMenuActions and AvailToolbarActions.

Location: toolbar.p

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

Notes: None

updateState

Procedure that receives state message events related to record updates.

Location: toolbar.p

Parameters:

INPUT pcState AS CHARACTER

Update state.

Notes: None

updateStates

Procedure that updates state temp-table whenever bands/actions are loaded. Adds actions to already-loaded states.

Location: toolbar.p

Parameters:

INPUT pcState AS CHARACTER

The name of some state. Empty string means **all**.

Notes: None

viewHideActions

Procedure that sets actions to visible or hidden according to state (called from setbuttons). Loops through the pcViewActions list, then the pcHideActions.

Location: toolbar.p

Parameters:

INPUT pcViewActions AS CHARACTER

List of actions to be made visible

INPUT pcHideActions AS CHARACTER

List of actions to be made HIDDEN

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

Procedure used to view an object. The purpose for the override is to add the menu bar handle to the window.

Location: toolbar.p

Parameters: None

Notes: None

Toolbar object properties

Toolbar Object properties provide information about toolbar objects and their classes. This information can include whether an object is enabled, the contents of the object and so on. You can read property values and in many instances you can change property values. To read a property value, you use a **get** function, and to change a property value, you use a **set** function.

These functions conform to the following conventions:

- **get** — Uses the form *getpropnam* and returns the current value of the property

NOTE: This function accepts no arguments.

- **set** — Uses the form *setpropname*. The set function accepts a single argument—the new value for the property—and returns TRUE/FALSE depending on whether the value change succeeds.

For more information about getting and setting property values, see [Chapter 1, “ADM2 SmartObject API Reference”](#) in this guide.

AvailMenuActions

Actions that are available in the menu of the toolbar object. Updated internally from insertMenu. The Instance Property dialog shows these and AvailToolbarActions. The actions selected are saved as ActionGroups.

Data Type: CHARACTER

Notes: Read and Write

AvailToolbarActions

List of actions available in the menu of this toolbar. The Instance Properties dialog shows these and AvailMenuActions. The actions selected are saved as ActionGroups.

Data Type: CHARACTER

Notes: Read and Write

AvailToolbarBands

Comma-separated list of the available toolbar bands for this toolbar master. Assembles the list by looping through the temp-table.

Data Type: CHARACTER

Notes: Read only

BoxRectangle

Handle to the rectangle, if any, around the buttons in the Panel used by `resizeObject`.

Data Type: HANDLE

Notes: Read only

BoxRectangle2

Used for bottom rectangle on toolbars where `ToolbarAutosize` is `TRUE`.

Data Type: HANDLE

Notes: Read and Write

ButtonCount

Number of buttons in a `SmartPanel`. Used by `resizeObject`.

Data Type: INTEGER

Notes: Read only

DeactivateTargetOnHide

Property used to control whether or not a target should be deactivated immediately on hide. If `TRUE`, the target should be deactivated.

Data Type: Logical

Notes:

- Read and Write
- If `TRUE`, the received event is returned to the `linkStateHandler` of the source procedure along with the handle of the toolbar so that it can activate or deactivate the link accordingly.
- If `FALSE`, an inactive state is ignored and an active event is used to pass **inactive** to the `linkstateHandler` procedure for all the targets that are not the source of the publish and **active** is passed to the source for activation.

DisabledActions

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.

Data Type: CHARACTER

Notes: Read and Write

EdgePixels

Value of EdgePixels.

Data Type: INTEGER

Notes: Read and Write

HiddenActions

Comma separated list of hidden actions.

Data Type: CHARACTER

Notes: Read and Write

HiddenMenuBands

Comma-separated list of hidden menu bands.

Data Type: CHARACTER

Notes: Read and Write

HiddenToolbarBands

Comma-separated list of hidden toolbar bands.

Data Type: CHARACTER

Notes: Read and Write

Image

Name of the button of interest or button-image number.

Data Type: LOGICAL

Notes: Write only

ImagePath

Path to the images in the file system.

Data Type: LOGICAL

Notes: Read and Write

MarginPixels

Number of pixels to reserve for the Panel margin used by resizeObject.

Data Type: INTEGER

Notes: Read only

Menu

Determines whether or not to generate a menu. TRUE if a menu is to be generated.

Data Type: LOGICAL

Notes: Read and Write

MenuMergeOrder

Determines the order in which menus are merged with other toolbar instances.

Data Type: INTEGER

Notes: Read and Write

MinHeight

Determines the minimum height.

Data Type: DECIMAL

Notes: Read and Write

- Read only in toolbar.p
- Read and Write in visual.p

MinWidth

Determines the minimum width.

Data Type: DECIMAL

Notes: Read only

NavigationTargetEvents

Comma-separated list of the events this object wants to subscribe to in its NavigationTarget.

Data Type: CHARACTER

Notes: Read only

NavigationTargetName

Object name of the Data Object to be navigated by this panel. This is set if the Navigation-Target is an SBO or other Container with DataObjects.

Data Type: CHARACTER

Notes: Read and Write

PanelFrame

Frame handle of the SmartPanel, for resizeObject.

Data Type: HANDLE

Notes: Read only

PanelLabel

Handle of the label for the Panel, if any.

Data Type: HANDLE

Notes: Read only

PanelState

Current state of the SmartPanel.

Data Type: CHARACTER

Notes: Read and Write

PanelType

Type of SmartPanel (**Update**, **Navigation**, and so on).

Data Type: CHARACTER

Notes: Read and Write

SecuredTokens

Comma-separated list of secured tokens available from the container.

Data Type: CHARACTER

Notes: Read and Write

ShowBorder

Determines whether or not a three-dimensional border is to be used around the buttons and as a delimiter when **RULE** is specified in createToolbar. TRUE if a three-dimensional border is to be used around.

Data Type: LOGICAL

Notes: Read and Write

StaticPrefix

Prefix used before the action name in static definitions. Needed for use with the Repository.

Data Type: CHARACTER

Notes: Read and Write

TableIOTarget

List of the handles, in character format, of the TableIO Targets of an object.

Data Type: CHARACTER

Notes: Read and Write

TableIOTargetEvents

Comma-separated list of the events to which this object wants to subscribe to in its TableIO Target.

Data Type: CHARACTER

Notes: Read and Write

TableIOType

Comma-separated list in CHARACTER format of the object's TableIO-Target handles.

TableIO-specific values of PanelType, if any, so that resetTabelio can handle panels similar to toolbar.

Data Type: CHARACTER

Notes: Read and Write

Toolbar

Determines whether or not to create a toolbar. TRUE if the toolbar is to be created.

Data Type: LOGICAL

Notes: Read and Write

ToolbarAutoSize

Determines whether or not the toolbar should be auto-sized to the width of the window at runtime. TRUE if the toolbar should be auto-sized.

Data Type: LOGICAL

Notes: Read and Write

ToolbarBands

List of the toolbar bands selected in the Instance Properties.

Data Type: CHARACTER

Notes: Read and Write

ToolbarDrawDirection

Draw direction: **horizontal** or **vertical** of the toolbar.

Data Type: CHARACTER)

Notes: Read and Write

ToolbarHeightPxI

Calculates and returns the toolbar height in pixels from the three properties: ToolHeightPXL, ToolbarMarginPxI, and ShowBorder.

Data Type: INTEGER

Notes: Read only

ToolbarInitialState

State selected in the Instance Properties. Used to view or hide buttons and menus according to state.

Data Type:

Notes: Read and Write

ToolbarTargetEvents

List of events to which this object wants to subscribe in the Toolbar-Target

Data Type: CHARACTER

Notes: Read and Write

ToolbarWidthPxI

Calculates and returns the toolbar width: ToolMarginPxI, ToolWidthPxI, and ToolMaxWidthPxI.

Data Type: INTEGER

Notes: Read only

ToolMarginPxI

Returns a constant zero.

Data Type: INTEGER

Notes: Read and Write

Action properties for toolbar objects

There are a number of action properties that you use to specify actions for the toolbar. All of these properties can be read and some of them can be set using **assign**. Action properties are class-level properties, and as a result, changing an action property using **assignAction<Property>** affects all toolbars that are initialized after the change.

[Table 6–1](#) lists and provides a brief description of the action properties for toolbar objects.

Table 6–1: Action properties for toolbar objects (1 of 6)

Action property	Description	Read	Write	Data type
Accelerator	Accelerator string, if any, for the specified action.	Yes	Yes	CHARACTER
AccessType	Access type, if any, for some action.	Yes	Yes	CHARACTER
AlternateImageRule	TRUE if able to successfully assign some value to the AlternateImageRule for some action.	No	Yes	CHARACTER

Table 6–1: Action properties for toolbar objects*(2 of 6)*

Action property	Description	Read	Write	Data type
Caption	Caption for some action, or the empty string if there is no caption defined.	Yes	Yes	CHARACTER
Category	Category for the action identified in the argument.	Yes	No	CHARACTER
Children	Comma-separated list of all children of a specified action.	Yes	No	CHARACTER
ControlType	Control-type for the action identified in the argument.	Yes	No	CHARACTER
CreateEvent	Creates an event for the specified action.	Yes	No	CHARACTER
Description	Value for the specified action.	Yes	Yes	CHARACTER
Disabled	Value of actionDisabled. TRUE if the action identified in the argument has been disabled.	Yes	No	CHARACTER
DisableRule	Flag indicating whether or not a value can be assigned to DisableRule. TRUE if a value can be assigned.	No	Yes	CHARACTER
EnableRule	The ImageEnableRule for a given action as stored in the repository.	Yes	Yes	CHARACTER

Table 6–1: Action properties for toolbar objects

(3 of 6)

Action property	Description	Read	Write	Data type
EnableRuleX	Query string for the specified argument. Valid values are reset , add , copy , delete , save , cancel , first , prev , next , and last .	Yes	No	
Event	Identifies the event to publish on Default-action of the browse. This property defines the persistent trigger that runs defaultAction and also subscribes the source-procedure.	Yes	Yes	CHARACTER
Groups	Identifies the action groups specified by Instance Properties. Instance Properties list the ADM instance properties of the selected SmartObject. See InstanceProperties for more information.	Yes	Yes	CHARACTER
HideRule	Value of HideRule for the action supplied as the argument.	Yes	Yes	CHARACTER
HideRuleX	A string value depending on the argument. The empty string is returned if the argument is not valid. Valid values are update , txtTableioOk , and txtTableioCancel .	Yes	No	CHARACTER
Image	Value for ActionImage	Yes	Yes	CHARACTER

Table 6–1: Action properties for toolbar objects

(4 of 6)

Action property	Description	Read	Write	Data type
ImageAlternate	Value actionImageAlternate.	Yes	No	CHARACTER
ImageAlternateRule	ImageAlternateRule for a given action as stored in the repository. The rules are evaluated in ruleStateChanges.	Yes	Yes	CHARACTER
InitCode	Value of InitCode.	Yes	No	CHARACTER
IsMenu	Flag determines whether or not an action is Menu. TRUE if the action is 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 does not appear as a selectable action in the instance property dialog even if actionIsParent returns a TRUE status.	Yes	No	CHARACTER
IsParent	Indicates whether or not an action is a parent. TRUE if the identified action is a parent.	Yes	No	CHARACTER
Label	Button label for the action button.	Yes	Yes	CHARACTER
Link	Link value for a specified action.	Yes	No	CHARACTER

Table 6–1: Action properties for toolbar objects*(5 of 6)*

Action property	Description	Read	Write	Data type
LogicalObjectName	LogicalObjectName value for a specified action.	Yes	No	CHARACTER
Name	Name value for a specified action.	Yes	Yes	CHARACTER
OnChoose	OnChoose value for a specified action.	Yes	No	CHARACTER
Order	Associates an integer order number with an action.	No	Yes	CHARACTER
Parameter	RunParameter value for a specified action.	Yes	No	CHARACTER
Parent	Parent value for a specified action.	Yes	Yes	CHARACTER
PhysicalObjectName	PhysicalObjectName value for a specified action.	Yes	No	CHARACTER
Refresh	Flag indicating whether or not the specified action can be refreshed. TRUE if the specified action can be refreshed.	Yes	No	CHARACTER
RunAttribute	RunAttribute value for a specified action.	Yes	No	CHARACTER
SecondImage	Image2 value for a specified action. TRUE if able to assign a new value to Image2 for some action.	Yes	Yes	CHARACTER
SecuredToken	SecurityToken value for a specified action.	Yes	No	CHARACTER

Table 6–1: Action properties for toolbar objects (6 of 6)

Action property	Description	Read	Write	Data type
Substitute Property	SubstituteProperty value for a specified action.	Yes	No	CHARACTER
Tooltip	Value depending on the value of TranslatedActionTooltip for a specified action. The values that might be returned are: TranslatedActionMLabel, TranslatedActionLabel, or Tooltip. TRUE if able to assign a new value to Tooltip for a specified action.	Yes	Yes	CHARACTER
Type	Type value for a specified action.	Yes	No	CHARACTER

Field Objects and Their Methods and Properties

This chapter lists and describes the methods (internal procedures and functions) and properties used for Toolbar Objects.

Field Object methods and properties let you define SmartObjects at a field level. Using these methods and properties, you can identify and get handles of fields and objects inside other objects for your application code. Refer to [Figure 1–1](#) to see the inheritance hierarchy for each object class.

NOTE: For information specific to the WebSpeed environment (see the “[Alphabetical Listing Of WebSpeed-specific API Routines](#)” chapter).

Base methods for field objects

The field methods support Smart technology at the field level. With these methods you can create SmartDataFields that support a wide variety of different visualizations.

initializeObject

The procedure generates special code for initializing SmartDataFields.

Location: field.p

Parameters: None

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.

resizeObject

Procedure that 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.

Location: field.p

Parameters:

INPUT pidHeight AS DECIMAL

New height.

INPUT pidWidth AS DECIMAL

New width.

Notes: No resizing is done by default. For this you need to create an override procedure in your SmartDataField with your own code, specific to your SmartDataField.

Methods for select field objects

You use the select methods for SmartSelect objects. These objects represent a self-populating selection list.

anyKey

Procedure that runs persistently on any key of the selection fill-in.

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().
```

Location: select.p

Parameters: None

browseHandler

Procedure that sets the properties in the newly started Browse widget.

Location: select.p

Parameters:

INPUT phBrowseObject AS HANDLE

Handle of the Browse.

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

buildList

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

Location: select.p

Parameters: None

Returns: CHARACTER PRIVATE

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

createLabel

Creates the label of the selection. The label is added to the parent frame.

Location: `select.p`

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.

dataAvailable

Procedure that defines logic to receive data from the data-source when the view-as is a fill-in.

Location: `select.p`

Parameters:

INPUT pcMode AS CHARACTER

Not used.

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.

destroyObject

Local override of destroyObject, this procedure deletes created widgets.

Location: `select.p`

Parameters: None

Notes: None

destroySelection

Cleans up all dynamically created objects.

Location: `select.p`

Parameters: None

Returns: LOGICAL PRIVATE

Notes: None

disableButton

Procedure that disables a SmartSelect's controls. Used to prevent finding and perhaps unintentionally changing a second record when an update is pending.

Location: `select.p`

Parameters: None

Notes: None

disableField

Procedure that disables SmartSelection component.

Location: `select.p`

Parameters: None

Notes: The SmartDataViewer container calls this procedure if a widget of type PROCEDURE is encountered in disableFields.

disable_UI

Procedure that disables the user interface parameters.

Location: `select.p`

Parameters: `None`

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.

enableButton

Procedure that undoes the action of `disableButton`.

Location: `select.p`

Parameters: `None`

Notes: `None`

enableField

Procedure that enables `SmartSelection` component.

Location: `select.p`

Parameters: `None`

Notes: The `SmartDataViewer Container` calls this procedure if a widget of type `PROCEDURE` is encountered in `enableFields`.

endMove

In design mode, this procedure moves the label when the field is moved.

Location: `select.p`

Parameters: `None`

Notes: Defined as `PERSISTENT` on `END-MOVE` of the frame of the widget.

enterSelect

When a trigger fires on entry of SmartSelect field, this procedure copies the screen value to the object. As a result, the lookup can be avoided if the new value actually exists as a unique record.

Location: `select.p`

Parameters: None

Notes: None

formattedValue

Returns the formatted value of a passed value according to the SmartSelects format.

Location: `select.p`

Parameters:

INPUT pcValue AS CHARACTER

The value that needs to be formatted.

Returns: CHARACTER

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).

hideObject

Procedure that overrides to ensure that the label is hidden

Location: `select.p`

Parameters: None

Notes: For sizing reasons, the label is not really a part of the object, but added as a text widget to the parent frame.

initializeBrowse

Code to initialize a browse. This procedure 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.

Location: `select.p`

Parameters: `None`

Notes: `None`

initializeObject

Procedure that overrides initializeObject in order to subscribe to the ChangedEvent, if defined, and to initializeSelection.

Location: `select.p`

Parameters: `None`

Notes: `None`

initializeSelection

Unless the view-as is specified to be a browse, this procedure creates a selection widget and populates it with data.

Location: `select.p`

Parameters: `None`

Notes: `None`

leaveSelect

This procedure fires on leave of a SmartSelect field and determines whether the lookup can be avoided because the new value exists as a unique record.

Location: `select.p`

Parameters:

Notes: This works only if the displayed field is in the first buffer of the linked SDO.

queryOpened

Procedure that subscribes to dataSource parameters.

Location: select.p

Parameters: None

Notes: SmartSelect needs to know that the query has changed, since this cannot be detected through an ordinary publish with dataAvailable.

refreshObject

Procedure that refreshes the data in a SmartSelect.

Location: select.p

Parameters: None

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.

repositionDataSource

Repositions the current dataSource according to the key field. Conditionally called from setDataValue() and valueChanged().

Location: select.p

Parameters:

INPUT pcValue AS CHARACTER

Returns: LOGICAL

Notes: None

resizeObject

Procedure that resizes the SmartSelect.

Location: select.p

Parameters:

INPUT pidHeight AS DECIMAL

New height of component.

INPUT pidWidth AS DECIMAL

New width of component.

Notes: The procedure deletes the current widget, resizes the frame, and re-creates the widget.

rowSelected

Procedure that publishes from the browser when a row is selected.

Location: select.p

Parameters: None

Notes: The browser is selected on default action and exits (if CancelrowObExit = FALSE). The subscribing is done in browseHandler.

valueChanged

Procedure used to notify the SmartDataViewer container that this value has changed.

Location: select.p

Parameters: None

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.

viewObject

Procedure that makes sure the label is viewed.

Location: select.p

Parameters: None

Notes: None

Base methods for combo and lookup objects

These methods support both combo and lookup objects. These methods support cache management, temp-tables, and AppServer requests.

notifyChildFields

This procedure refreshes child combos. The procedure calls one or more APIs to run on the target depending on the type of event specified in the input parameter. The following table lists the APIs that run in response to each value:

pcEvent value	APIs run
Prepare	prepareField
Display	displayfield
Retrieve	prepareField retrieveData
Fetch	preparefield retrieveData displayField

Location: lookupField.p

Parameters:

INPUT pcEvent AS CHARACTER

The type of event for which to process. Allowed values are “Prepare”, “Display”, “Retrieve”, and “Fetch”.

Notes:

- This procedure replaces the refreshChildDependancies procedure. The refreshChildDependancies procedure exists for backwards compatibility and might be fully deprecated in the future.
- A lookup's children and it's linked fields' children are not currently refreshed by this procedure.

retrieveData

This procedure retrieves data for the combos and lookup on a viewer.

Location: lookupField.p

Parameters:

INPUT phViewer AS HANDLE

 The viewer handle.

Notes: None

returnParentFieldValues

This procedure resolves the ParentFilter of a combo or lookup and appends the filter subclause to the main query.

Location: lookupField.p

Parameters:

OUTPUT pcNewQuery AS CHARACTER

 The main query with the filtering subclauses.

Notes: This procedure replaces the procedure of the same name that existed in the Lookup and Combo classes in previous versions. For backwards compatibility, this procedure calls a copy of the old API when use of the legacy APIs has been specified with the keep_old_field_API session property.

Methods for combo field objects

The following section lists and describes the methods for SmartCombo field objects.

anyKey

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

Location: `combo.p`

Parameters: None

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

createDataSource

When an DynCombo uses a DataObject rather than the application database, this procedure starts the defined DataObject as an instance in the container and add the data link from the DataObject to the DynCombo.

Location: `combo.p`

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

Creates the label for the Combo Box.

Location: `combo.p`

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.

destroyCombo

Cleans up all dynamically created objects.

Location: combo.p

Parameters: None

Returns: LOGICAL

Notes: None

destroyObject

Local override of destroyObject. This procedure deletes dynamic Combo Box objects by calling destroyCombo.

Location: combo.p

Parameters: None

Notes: None

disableField

Procedure that sets SENSITIVE and FieldEnabled to FALSE.

Location: combo.p

Parameters: None

Notes: The SmartDataViewer container calls this procedure if a widget of type PROCEDURE is encountered in disableFields.

disable_UI

Procedure that purges THIS-PROCEDURE, if persistent.

Location: combo.p

Parameters: None

Notes: Clean up the user-interface. This routine is usually called after execution completes.

displayField

This procedure manages the display of the object's current value.

Location: combo.p

Parameters: None

Notes: This procedure replaces the displayCombo procedure. The displayCombo procedure exists for backwards compatibility and might be fully deprecated in the future.

displayCombo

This procedure 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 ComboQuery.

Location: combo.p

Parameters:

INPUT TABLE FOR ttDCombo

Notes:

- This procedure has been replaced with the displayField procedure. This procedure exists for backwards compatibility and might be fully deprecated in the future.
- This is designed to allow all combo queries to be built at the cost of only a single appserver call.

enableField

Procedure that enables a Combo field. The SmartDataViewer calls this procedure if a widget of type PROCEDURE is encountered in enableFields.

Location: combo.p

Parameters: None

Notes: None

enterCombo

This procedure fires a trigger on entry into a Combo field to store the current screen value of the displayed field.

Location: combo.p

Parameters: None

Notes: The typical case is that the displayed field are not the same as the keyfield. In that case, logic can be fired 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.

endMove

This procedure moves the label whenever the field is moved in design mode.

Location: combo.p

Parameters: None

Notes: Defined as PERSISTENT on END-MOVE of the frame of the widget.

hideObject

This procedure gets the value of LabelHandle, hides the label, and then calls RUN SUPER.

Location: combo.p

Parameters: None

Notes: For sizing reasons, the label is not really a part of the object, but added as a text widget to the parent frame.

initializeCombo

Procedure that runs as part of initializeObject to set up the combo.

Location: combo.p

Parameters: None

Notes: Defaults keyfield to actual field if not defined. Defaults displayed field to keyfield if not defined. Assumes a data type of decimal for keyfield and character for displayed field if different.

insertExpression

Inserts an expression into ONE buffer's WHERE clause.

Location: combo.p

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.

leaveCombo

This procedure fires a trigger on leave of combo field. It stores the current screen value in the combo.

Location: combo.p

Parameters: None

Notes: None

newQueryString

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 is always embedded in parenthesis.

Location: `combo.p`

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.

newWhereClause

Inserts a new expression to query's prepare string for a specified buffer.

Location: combo.p

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 does not use properties.

prepareField

Prepares the field for retrieval by building the query with current parent values added to the baseQueryString.

Location: combo.p

Parameters: None

Notes: This procedure replaces getComboQuery. The getComboQuery procedure exists for backwards compatibility but might be fully deprecated in the future.

refreshChildDependancies

Procedure that refreshes any children of the Combo identified in the argument.

Location: combo.p

Parameters:

INPUT pcFieldName AS CHARACTER

The possible parent.

Notes: This procedure has been replaced by the notifyChildFields procedure in the lookupField class. This procedure exists for backwards compatibility but might be fully deprecated in the future

refreshCombo

This procedure runs displayCombo in TARGET-PROCEDURE. This is usually triggered from a value-changed event of a parent combo.

Location: combo.p

Parameters: None

Notes: None

resizeObject

This procedure resizes the Combo by deleting the widget, resizing the frame, and recreating the widget in the new size.

Location: combo.p

Parameters:

INPUT pidHeight AS DECIMAL

New height of the widget.

INPUT pidWidth AS DECIMAL

New width of the widget.

Notes: None

returnParentFieldValues

This procedure loops through all field handles.

Location: combo.p

Parameters:

OUTPUT pcNewQuery AS CHARACTER

Notes: None

showQuery

Procedure that displays the current query of the combo.

Location: combo.p

Parameters: None

Notes: None

valueChanged

Procedure used to notify the SmartDataViewer that the Combo's value has changed. Publishes populateChildCombo and comboValueChanged.

Location: combo.p

Parameters: None

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.

whereClauseBuffer

Returns the buffername of a WHERE clause expression. This function avoids problems with leading or double blanks in WHERE clauses.

Location: combo.p

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.

viewObject

Procedure that turns off HIDDEN to make the label visible.

Location: combo.p

Parameters: None

Notes: For sizing reasons, the label is not really a part of the object, but added as a text widget to the parent frame.

Methods for lookup field objects

The following sections lists and describes the methods for SmartLookup field objects. A SmartLookup is a faster but less general version of a SmartSelect object.

anyKey

Procedure used to trap a keypress and display a single-value return by going directly to the query.

Location: lookup.p

Parameters: None

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

createLabel

Creates the label for the Lookup.

Location: lookup.p

Parameters:

INPUT pcLabel AS CHARACTER

 The label string.

Returns: HANDLE

This function is separated in order to create the label in design-mode. The label is defined in the parent Frame.

destroyLookup

Destroys associated lookups, labels, and buttons.

Location: lookup.p

Parameters: None

Returns: LOGICAL

Notes: None

destroyObject

Local override of destroyObject, this procedure deletes created widgets.

Location: lookup.p

Parameters: None

Notes: None

disableButton

Procedure that disables the specific Lookup's button and Browser to prevent the use of the Lookup.

Location: lookup.p

Parameters: None

Notes: Use this procedure when you want to modify a specific record, but do not want a user to use a specific Lookup to find another record and potentially modify the record.

disableField

Procedure that disables Lookup field in a Viewer.

Location: lookup.p

Parameters: None

Notes: The SmartDataViewer calls this procedure if a widget of type PROCEDURE is encountered in disableFields.

disable_UI

Procedure that disables the user interface and releases memory, typically at the end of a session.

Location: lookup.p

Parameters: None

Notes: This procedure cleans up the user interface by deleting dynamic widgets that were created and hiding Frames.

displayField

This procedure manages the display of the object's current value.

Location: lookup.p

Parameters: None

Notes: This procedure replaces the displayCombo procedure. The displayCombo procedure exists for backwards compatibility and might be fully deprecated in the future.

displayLookup

Procedure that 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.

Location: lookup.p

Parameters:

INPUT TABLE FOR ttLookup

Notes:

- This procedure has been replaced with the displayField procedure. This procedure exists for backwards compatibility and might be fully deprecated in the future.

- 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.

enableButton

If you disable the Lookup button to prevent the user from using it, this procedure calls this routine to re-enable the button again.

Location: lookup.p

Parameters: None

Notes: None

enableField

This procedure, enables a Lookup Field. The SmartDataViewer calls this routine if a widget of type PROCEDURE is encountered in enableFields.

Location: lookup.p

Parameters: None

Notes: None

endMove

In design mode, this procedure moves the label when the field is moved.

Location: lookup.p

Parameters: None

Notes: Defined as PERSISTENT on END-MOVE of the frame of the widget.

enterLookup

This procedure fires a trigger on entry of lookup field.

Location: lookup.p

Parameters: None

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.

hideObject

This procedure gets the value of LabelHandle, hides the label, and then calls RUN SUPER.

Location: lookup.p

Parameters: None

Notes: For sizing reasons, the label is not really a part of the object, but added as a text widget to the parent frame.

initializeBrowse

This procedure initializes the lookup browser window.

Location: lookup.p

Parameters: None

Notes: None

initializeLookup

This procedure runs as part of initializeObject to set up the lookup.

Location: lookup.p

Parameters: None

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.

insertExpression

PRIVATE function. Inserts an expression into one buffer's WHERE clause.

Location: lookup.p

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.

leaveLookup

This procedure is fired on leave of a lookup field and handles the situation when a value is manually entered rather than using the lookup key.

When this occurs, it must be determined whether the screen value has changed. If the screen value has changed, it must be verified whether the changed screen value is a valid record. If it is a valid record, the key value is reset to avoid using the lookup

Location: lookup.p

Parameters: None

Notes: If you override this procedure, you should include a “RETURN NO-APPLY” statement somewhere after the “RUN SUPER” statement.

newQueryString

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 is always enclosed in parentheses.

Location: `lookup.p`

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 datatype.

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.

newWhereClause

Appends a new expression to the query's prepare string for a specified buffer.

Location: lookup.p

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

NOTE: This is a utility function that does not use properties.

prepareField

Prepares the field for retrieval by building the query with current parent values added to the baseQueryString.

Location: lookup.p

Parameters: None

Notes: This procedure replaces getLookupQuery. The getLookupQuery procedure exists for backwards compatibility but might be fully deprecated in the future.

resizeObject

This procedure resizes the Lookup by deleting the widget, resizing the Frame, and recreating the widget.

Location: lookup.p

Parameters:

INPUT pidHeight AS DECIMAL

 The new height.

INPUT pidWidth AS DECIMAL

 The new width.

Notes: None

returnParentFieldValues

Value of returnParentFieldValues.

Location: lookup.p

Parameters:

OUTPUT pcNewQuery AS CHARACTER

Notes: None

rowSelected

This procedure publishes a lookup complete event in the container allowing developers to perform some action when a row is selected from the browser.

Location: lookup.p

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.)

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.

translateBrowseColumns

This procedure translates lookup and filter browse columns.

Location: lookup.p

Parameters:

INPUT pcObjectName AS CHARACTER

INPUT phBrowseHandle AS HANDLE

Notes: None

valueChanged

This procedure sets DataModified to YES to ensure that the SmartDataViewer container is notified that the value has changed for Lookup.

Location: lookup.p

Parameters: None

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.

viewObjec

This procedure makes sure the label is not HIDDEN.

Location: lookup.p

Parameters: None

Notes: For sizing reasons, the label is not really a part of the object, but added as a text widget to the parent frame.

whereClauseBuffer

Returns the buffer name of a WHERE clause expression. This function avoids problems with leading/double blanks in WHERE clauses.

Location: lookup.p

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.

Field object properties

Field object properties provide information about field objects and its classes. This information can include whether an object is enabled, the contents of the object and so on. You can read property values and in many instances you can change property values. To read a value for a property, you use a **get** function, and to change a value for a property, you use a **set** function.

These functions conform to the following conventions:

- **get** — Uses the form *getpropnam* and returns the current value of the property

NOTE: This function accepts no arguments.

- **set** — Uses the form *setpropname*. The set function accepts a single argument—the new value for the property—and returns TRUE/FALSE depending on whether the value change succeeds.

For more information about getting and setting property values, see [Chapter 1, “ADM2 SmartObject API Reference”](#) in this guide.

The following section lists and describes the Field object properties that you can use with the get and set functions. The description also identifies the properties for which you can read and write (change) a value and the properties for which you can only read the value.

BaseQueryString

Base query string used for dynamic Lookups and dynamic Combos. This is the original query assigned to the object when initialized.

Data Type: CHARACTER

Notes: Read and Write

BrowseFields

Comma-separated list of field names to display in the lookup browse for the dynamic Lookup.

Data Type: CHARACTER

Notes: Read and Write

BrowseFieldFormats

Pipe-delimited list of field formats that correspond to the BrowseFields properties. These formats are assigned to the fields in the browser being used for the dynamic Lookup browserValue of BrowseFieldFormats.

Data Type: CHARACTER

Notes: Read and Write

BrowseFields

List of fields to display in the browse when the viewAs property is set to **browse**.

Data Type: CHARACTER

Notes: Read and Write

BrowseTitle

Title to display in the browse SmartWindowContainer when the ViewAs property is set to **browse**.

Data Type: CHARACTER

Notes: Read and Write

BuildSequence

Used for dynamic Combos to determine the order in which to initialize a dynamic Lookup. This property allows you to use of the Parent filter query option when you have a dynamic Combo that is a parent for another dynamic Combo.

Data Type: INTEGER

Notes: Read and Write

CancelBrowseOnExit

Controls whether or not to select the value in the browse on exit. If TRUE, the value in the browse is NOT to be selected on exit.

Data Type: LOGICAL

Notes: Read and Write

ChangedEvent

Optional event to publish on value-changed.

Data Type: CHARACTER

Notes: Read and Write

ComboBuffer

Returns a buffer to the ttDCombo temp-table.

Data Type: HANDLE

Notes: Read

ComboDelimiter

Property assigned at runtime that contains the delimiter assigned to a dynamic Combo.

Data Type: CHARACTER

Notes: Read and Write

ComboFlag

Property that allows you to specify options for a dynamic Combo. You can specify the following for this property:

- **Blank** — If a dynamic Combo should only contain the data retrieved from the BaseQueryString.
- **A** — If the dynamic combo allows the user to select a value of <All>.
- **N** — If the dynamic combo allows the user to select a value of <None>.

If the value for this property is **A** or **N**, the ComboFlagValue property must also have a value.

Data Type:

Notes: Read and Write

ComboFlagValue

The value for this property is used to assign a value to a record when the user selects either **All** or **None** for a dynamic combo. This property contains a value only when the ComboFlag property has a value of **A** or **N**.

Data Type: CHARACTER

Notes: Read and Write

ComboHandle

The value for this property is available only at run time and contains the handle of the combo-box of a dynamic Combo.

Data Type: HANDLE

Notes: Read and Write

ComboQuery

Used to pass the query required by this combo back to the viewer for building. Once built, the query is returned into the procedure displayCombo.

Data Type:

Notes:

- This function has been replaced by the prepareField API. It remains for backward compatibility and might be fully deprecated in the future.
- Read only
- This design facilitates all combo queries being built with a single AppServer hit. It is published in the viewer.
- This function is not run in add mode.
- This function is published from displayFields in the viewer.

ComboSort

Sort property for the Combo.

Data Type: LOGICAL

Notes: Write only

CurrentDescValue

The value for this property is available only at run time and contains a description of the currently selected option in a dynamic Combo. Basically, this is the current SCREEN-VALUE of the dynamic Combo.

Data Type: CHARACTER

Notes: Read and Write

CurrentKeyValue

The value for this property is available only at run time and contains the current key value for the selected option. The value of this property is the same as getting the DataValue of the dynamic Combo.

Data Type: CHARACTER

Notes: Read and Write

DataModified

Indicates whether SmartDataField value has changed but not saved. If TRUE, the field has changed and saved. If FALSE, the field has changed but not saved.

Data Type: LOGICAL

Notes: Read and Write

DataSourceFilter

This property is an optional filter expression for the data source.

Data Type: CHARACTER

Notes: Read and Write

DataSourceName

This property is the name of the SDO that a Dynamic Combo is using as a data source instead of using the application database.

Data Type: CHARACTER

Notes: Read and Write

DataValue

This property is used by the SmartDataViewer when it collect changes using collectChanges.

Data Type: CHARACTER

Notes:

- Read and Write
- If it encounters this PROCEDURE in the list of EnabledHandles and the DataModified property of this object equals TRUE.

DefineAnyKeyTrigger

Controls whether or not a persistent trigger is to be defined on ANY-KEY. TRUE if a persistent trigger is defined. Does not use the {get} syntax

Data Type: LOGICAL

Notes: Read and Write

DescSubstitute

Property that stores the sequence for displaying the fields specified in the DisplayedField property list in a dynamic Combo. For example, &1 (&2) displays the value of the first field specified in DisplayedField and then concatenates that value with the value of the second field.

Data Type: CHARACTER

Notes: Read and Write

DisplayDataType

Property that stores the data type of DisplayField for both the dynamic Lookup and dynamic Combo.

Data Type: CHARACTER

Notes: Read and Write

DisplayField

Determines whether or not the SmartDataField is to be displayed along with other fields in its Contain. If TRUE, the SmartDataField is to be displayed. If FALSE, the SmartDataField is not to be displayed.

Data Type: CHARACTER

Notes: Read and Write

DisplayFormat

This property stores the format for DisplayedField for both the Dynamics Lookup and Dynamics Combo.

Data Type: CHARACTER

Notes: Read and Write

DisplayValue

Saved screen or display value of a SmartDataField.

Display Value 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 displayedfield value that is obtained from the SDO. In all other cases it searches the list-item-pairs (or radio-buttons for radio-sets) given the datavalue and will return the display value.

Data Type: CHARACTER

Notes: Read and Write

EdgePixels

Number of pixels that should be used to draw the rectangle around the buttons on a SmartPanel.

Data Type: INTEGER

Notes: Read and Write

EnableField

Determines whether or not the SmartDataField is to be enabled for user input along with other fields in its Container. TRUE if the SmartDataField is to be enabled for user input along with other fields in its Container, otherwise FALSE.

Data Type: CHARACTER

Notes: Read and Write

EnableOnAdd

Value of EnableOnAdd.

Data Type: LOGICAL

Notes: Read and Write

ExitBrowseOnAction

Determines whether or not the selection of a value in the browse should also Exit the browse. TRUE if the selection of a value in the browse should Exit.

Data Type: LOGICAL

Notes: Read and Write

FieldEnabled

Determines whether or not the SmartDataField is enabled for user input. TRUE if the SmartDataField is enabled, otherwise FALSE.

Data Type: LOGICAL

Notes: Read and Write

FieldHidden

Hides or unhides a lookup field.

Data Type:

Notes: Read only

FieldLabel

Label to use for the specified field.

Data Type: CHARACTER

Notes: Read and Write

FieldName

Name of the SDO field to which this object maps.

Data Type: CHARACTER

Notes: Read and Write

FieldToolTip

Text used for a tool tip for the specified field.

Data Type: CHARACTER

Notes: Read and Write

FlagValue

Option flag key values for **All** and **None**.

Data Type: CHARACTER

Notes: Read and Write

Format

An overridden format string.

Data Type: CHARACTER

Notes: Read and Write

HelpId

Optional HelpId of the selection.

Data Type: INTEGER

Notes: Read and Write

InnerLines

Option flag key values for **All** and **None**.

Data Type: INTEGER

Notes: Read and Write

KeyDataType

Property that stores the data type for the KeyField for both the dynamic Lookup and dynamic Combo.

Data Type: CHARACTER

Notes: Read and Write

KeyField

Name of the field whose value is either received or retrieved from the SmartDataViewer using the value in DataValue. For more information, see DataValue.

Data Type: CHARACTER

Notes: Read and Write

KeyFormat

Property that stores the KeyField format for both the dynamic Lookup and dynamic Combo.

Data Type: CHARACTER

Notes: Read and Write

Label

Label defined for the selection. If the value is undefined, it is converted to a string.

Data Type: CHARACTER

Notes: Read and Write

LabelHandle

Property value that is available only at run time and contains the handle of the lookup FILL-IN's handle of a dynamic Lookup.

Data Type: HANDLE

Notes: Read and Write

LinkedFieldDataTypes

Property that identifies the Data Types of Linked Fields to display in a viewer when a user selects a value in a dynamic Lookup. These values are comma-separated.

Data Type: CHARACTER

Notes: Read and Write

LinkedFieldFormats

Property that identifies the Formats of Linked Fields to display in viewer when a user selects a value in a dynamic Lookup. These values are pipe-delimited.

Data Type: CHARACTER

Notes: Read and Write

ListItemPairs

Property value that is available only at run time and contains the current list-item-pairs of a selected dynamic Combo.

Data Type: CHARACTER

Notes: Read and Write

LookupBuffer

Returns a buffer to the ttLookup temp-table.

Data Type: HANDLE

Notes: Read

LookupFilterValue

Property value that is valid only at run time and contains the last value of a dynamic Lookup when the user presses the lookup button or the shortcut-key. You use this value to filter the values returned in the lookup browse.

Data Type: CHARACTER

Notes: Read and Write

LookupHandle

Property value that is available only at run time and contains the handle of the FILL-IN of a dynamic Lookup.

Data Type: HANDLE

Notes: Read and Write

LookupImage

Property that contains a relative-path name of the image to display on the lookup button.

Data Type: CHARACTER

Notes: Read and Write

LookupQuery

Passes the query required by this lookup back to the viewer for building. Once built, the query is returned into the procedure displayLookup.

Data Type:

Notes:

- This function has been replaced by the prepareField API. It remains for backward compatibility and might be fully deprecated in the future.
- Read only
- The query is published from displayfields in the viewer to facilitate all lookup queries being built with a single AppServer hit.
- This routine is not run at all in add mode.

MaintenanceObject

Represents the logical object name for the lookup to be launched when allowing maintenance.

Data Type: CHARACTER

Notes: Read and Write

MaintenanceSDO

Name of the SDO to launch when allowing maintenance for the lookup.

Data Type: CHARACTER

Notes: Read and Write

NumRows

Number of rows to display in the selection widget.

Data Type: INTEGER

Notes: Read and Write

Optional

Indicates whether or not the selection is optional, and as a result, the property OptionalString holds the value to display. TRUE if the selection is optional and the property OptionalString holds the value to display.

Data Type: LOGICAL

Notes: Read and Write

OptionalBlank

Determines whether or not the optional value is a blank value. TRUE if the optional value is a blank value.

Data Type: LOGICAL

Notes:

- Read and Write
- Applies to character fields only.

OptionalString

Displays an optional value when the Optional property is set to TRUE.

Data Type: CHARACTER

Notes: Read and Write

ParentField

Parent field name of the parent dependent object of a Combo or Lookup.

Data Type: CHARACTER

Notes: Read and Write

ParentFilterQuery

Foreign fields of the parent object.

Data Type: CHARACTER

Notes: Read and Write

PopupOnAmbiguous

Property used to determine whether or not a Lookup Browse should automatically open when the results of a search string are ambiguous.

Use this property to open a Lookup Browse when the search string is modified and a match cannot be found because the search string is ambiguous. This is the default behavior.

Data Type: LOGICAL

Notes:

- Read and Write
- Not available in Progress Dynamics Web clients.

PopupOnUniqueAmbiguous

Property used to determine whether or not a Lookup Browse should automatically open when the results of a search string include multiple matches that are not unique but ambiguous.

Use this property to open a Lookup Browse when the search string returns multiple matches that are unique but ambiguous. For example, if you search for “John”, all instances of “John” are found, but instances that begin with “John”, such as “Johnson”, are also found.

Data Type: LOGICAL

Notes:

- Read and Write
- Not available in Progress Dynamics Web clients.

PopupOnNotAvail

Property used to determine whether or not a Lookup Browse should automatically open when an exact match cannot be found for a search string.

Use the property to open a Lookup Browse when an exact match cannot be found.

Data Type: LOGICAL

Notes:

- Read and Write
- Not available in Progress Dynamics Web clients.

QueryTables

Property that contains a comma-separated list of tables used in the BaseQueryString of a dynamic Lookup or Combo.

Data Type: CHARACTER

Notes: Read and Write

RefreshList

Comma-separated list of the names of the Dynamic Combos to be refreshed for the parent Combo named in the argument.

Data Type: CHARACTER

Notes: Read only

RepositionDataSource

Determines whether or not the data-source is to be repositioned on valueChanged of the select. TRUE if the data-source is to be repositioned.

Data Type: LOGICAL

Notes:

- Read and Write
- 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.

RowsToBatch

Number of records read in a single operation.

Data Type: INTEGER

Notes: Read and Write

SavedScreenValue

Replaced by DisplayValue (which it calls). Property is used for backwards compatibility.

Data Type: LOGICAL

Notes: Read and Write

SDFFileName

Property that contains the name of the SmartObject for the Lookup or Combo when you save a dynamic Lookup or Combo to the Repository. You can use this property to identify the object in the Repository when you want to make modifications.

Data Type: CHARACTER

Notes: Read and Write

SDFTemplate

Property used to save the name of the SmartDataField (SDF) when a dynamic Combo or dynamic Lookup is created using an existing SDF from the Repository.

Data Type: CHARACTER

Notes: Read and Write

Secured

Determines whether or not the combo's security is set to HIDDEN. TRUE, if the combo's security is set to HIDDEN.

Data Type: LOGICAL

Notes: Read and Write

StartBrowseKeys

List of Keylabels or KeyFunctions that starts the browse.

Data Type: CHARACTER

Notes: Read and Write

ToolTip

Optional ToolTip for the selection.

Data Type: CHARACTER

Notes: Read and Write

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.

Data Type: LOGICAL

Notes: Read only

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.

Data Type: CHARACTER

Notes: Read and Write

ViewerLinkedFields

A comma-separated list of fields that you want returned when a value is selected in a dynamic Lookup. You can then use these values either to fill selected fields on a viewer or as references to values required by the developer.

Data Type: CHARACTER

Notes: Read and Write

ViewerLinkedWidgets

If the ViewerLinkedFields attribute contains a value and the values returned from these fields should be automatically filled on a viewer when a value was selected in a dynamic Lookup, then this attribute would contain a comma-separated list of widget names on the viewer where these values should be filled in.

Data Type: CHARACTER

Notes: Read and Write

Field object column properties

There are a number of column properties available for which you can obtain and set (write) field values. All of these properties can be read and some of them can be set. To read and set column properties, you use the following prefixes with the specific column property:

- **Column** — Use to read the value of a specific column property. For example, if you want to read the value of the Format property, you would specify **Format**. This would return the format of the browse column you specify.
- **Assign** — Use to set the value of the specified column property.

NOTE: For Container objects, there are no column properties for which you can assign the value.

For additional information, see the *Progress Dynamics Programming Handbook*.

[Table 7–1](#) lists the column properties, provides a brief description of the property, and indicates whether the property can be read and set (Write).

Table 7–1: Column properties for field objects

Column property	Description	Read	Write
Format	Browse column format overrides	Yes	Yes
Labels	Browse column label overrides values.	Yes	Yes

Messaging Objects and Their Methods and Properties

This chapter lists and describes the methods (internal procedures and functions) and properties used for Messaging Objects. Refer to [Figure 1–1](#) to see the inheritance hierarchy for each object class.

NOTE: For information specific to the WebSpeed environment (see the [“Alphabetical Listing Of WebSpeed-specific API Routines”](#) chapter).

Base methods for messaging objects

This section lists and describes base methods for Messaging objects.

destroyObject

Procedure that deletes the JMS session if no other messaging object instances are using it. This routine is used by all messaging objects.

Location: messaging.p

Parameters: None

Notes: None

errorHandler

Handles asynchronous errors, if any.

Location: messaging.p

Parameters:

INPUT phMessage AS HANDLE

INPUT phMessageConsumer AS HANDLE

OUTPUT phReply AS HANDLE

Notes: This adds the message to the message queue for later retrieval.

initializeObject

Procedure that initializes objects of class Messaging.

Location: messaging.p

Parameters: None

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.

Methods for consumer messaging objects

This section lists and describes methods for consumer Messaging objects.

assignUnsubscribe

Set the value for the unsubscribe-on-close flag for a persistent subscription. You use this function when you want an application to prompt users at log off whether they want to cancel a subscription that would otherwise persist into the next session

Location: `consumer.p`

Parameters:

INPUT `pcDestination` AS CHARACTER

INPUT `pcSubscription` AS CHARACTER

INPUT `p1Unsubscribe` AS LOGICAL

createConsumers

Procedure that creates message consumers for normal message retrieval, shutdown message retrieval, and error handling.

Location: `consumer.p`

Parameters: None

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.

defineDestination

Creates a tDestination temp-table record for use by createConsumers() for creating message consumers.

Location: consumer.p

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.

destroyObject

The message consumer's version of destroyObject, this procedure deletes the message consumers and cancels any durable subscriptions that need to be cancelled.

Location: consumer.p

Parameters: None

Notes: None

errorHandler

The SmartConsumer's version of initializeObject, this procedure fetches messages to get and display from the SUPER version of ErrorHandler.

Location: consumer.p

Parameters:

INPUT phMessage AS HANDLE

INPUT phMessageConsumer AS HANDLE

OUTPUT phReply AS HANDLE

Notes: None

initializeObject

Procedure that is a SmartMessageConsumer version of initializeObject.

Location: consumer.p

Parameters: None

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.

messageHandler

This procedure runs when a message is received and handles the incoming message by calling appropriate procedures in its INMESSAGE-TARGET.

Location: consumer.p

Parameters:

INPUT phMessage AS HANDLE

INPUT phMessageConsumer AS HANDLE

OUTPUT phReply AS HANDLE

Notes: None

processDestinations

Procedure that creates tDestination temp-table records from the Destinations, Subscriptions, and Selectors property values.

Location: consumer.p

Parameters: None

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.

startWaitFor

Procedure that starts watching for messages when in batch mode.

Location: `consumer.p`

Parameters: None

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

Procedure that clears the `Waiting` property to shut down the `Consumer` object.

Location: `consumer.p`

Parameters:

INPUT `phMessage` AS HANDLE

INPUT `phMessageConsumer` AS HANDLE

OUTPUT `phReply` AS HANDLE

Notes: The `stopHandler()` routine is used to handle a `ShutDown` message, typically when in batch mode.

Methods for producer messaging objects

The following section lists and describes the methods for producer Messaging objects.

destroyObject

Procedure that deletes the `Reply` and `Error` message consumers. `SmartProducer`'s version of `destroyObject`.

Location: `producer.p`

Parameters: None

Notes: None

initializeObject

Procedure that initializes objects of class Producer.

Location: producer.p

Parameters: None

Notes: None

replyHandler

Procedure that handles replies to messages.

Location: producer.p

Parameters:

INPUT phReply AS HANDLE

INPUT phConsumer AS HANDLE

OUTPUT phResponse AS HANDLE

Notes: None

sendMessage

Procedure that creates and sends a message.

Location: producer.p

Parameters:

INPUT pcDestination AS CHARACTER

INPUT plReplyRequired AS LOGICAL

INPUT pcReplySelector AS CHARACTER

OUTPUT pcMessageID AS CHARACTER

Notes: This is run from the InMessage-Source to send a message.

Method for message handler objects

The following section lists and describes the method for Message-handler objects.

sendMessage

Procedure that sends the message to the OutMessage-Target.

Location: msghandler.p

Parameters: None

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.

Methods for XML messaging objects

The following section lists and describes methods for XML Messaging objects.

assignAttribute

pdOwner node and *pcValue* for the attribute *pcName*.

Location: xml.p

Parameters:

INPUT *pdOwner* AS DECIMAL

INPUT *pcName* AS CHARACTER

INPUT *pcValue* AS CHARACTER

Returns: LOGICAL

Notes: None

assignNodeValue

Value of a node.

Location: xml.p

Parameters:

INPUT pdNode AS DECIMAL

INPUT pcValue AS CHARACTER

Returns: LOGICAL

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

createAttribute

Creates an Attribute in an element and returns its unique id.

Location: xml.p

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.

createDocument

Creates an empty document.

Location: xml.p

Parameters: None

Returns: LOGICAL

Notes: None

createElement

Creates an Element in the document and returns its unique id.

Location: `xml.p`

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

createNode

Creates a Node in the document and returns its id.

Location: `xml.p`

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

createText

Creates a Text Node in the document and returns its unique id.

Location: xml.p

Parameters:

INPUT pdParent AS DECIMAL

 NodeId of Parent Element.

INPUT pcText AS CHARACTER

Returns: DECIMAL

Notes: None

deleteDocument

Deletes the document and cleans up all the temp-tables.

Location: xml.p

Parameters: None

Returns: LOGICAL

Notes: None

destroyObject

Procedure that destroys the document. Override this routine to delete the document and temp-table.

Location: xml.p

Parameters: None

Notes: None

nodeHandle

Returns the handle of the node identified by the `pdId` argument.

Location: `xml.p`

Parameters:

INPUT `pdId` AS DECIMAL

Returns: CHARACTER

Notes: None

nodeType

Returns the type of the node identified by *pdId*.

Location: `xml.p`

Parameters:

INPUT `pdId` AS DECIMAL

Returns: CHARACTER

Notes: None

ownerElement

Returns the OwnerElement corresponding to `pdAttributeNode`.

Location: `xml.p`

Parameters:

INPUT `pdAttributeNode` AS DECIMAL

Returns: DECIMAL

parentNode

Returns the ID of *pdNode*'s parent.

Location: `xml.p`

Parameters:

INPUT `pdNode` AS DECIMAL

ID of the node whose parent is wanted.

Returns: DECIMAL node ID of the parent.

Notes: None

processCDATASection

Procedure that processes a CDATA-Section node.

Location: xml.p

Parameters:

INPUT phText AS HANDLE

INPUT pcPath AS CHARACTER

Notes: This method just passes an event with path and value information.

processComment

Procedure that processes a comment node.

Location: xml.p

Parameters:

INPUT phText AS HANDLE

INPUT pcPath AS CHARACTER

Notes: This method does nothing and must be overridden if you need to process comments.

processDocument

Procedure that starts processing of the current document.

Location: xml.p

Parameters: None

Notes: None

processElement

Procedure that processes an element document.

Location: xml.p

Parameters:

INPUT phNode AS HANDLE

Current x-noderef handle.

INPUT pcPath AS CHARACTER

Current Document path (This is the logical path and not the numbered XPath).

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.

processRoot

Procedure that starts processing the elements of the document.

Location: xml.p

Parameters: None

This is just a starting point without events.

processText

Procedure that processes a text node.

Location: xml.p

Parameters:

INPUT phText AS HANDLE

INPUT pcPath AS CHAR

Notes: Passes the event with path and value information.

receiveHandler

Procedure that handles receiving a message.

Location: `xml.p`

Parameters:

INPUT `phMessage` AS HANDLE

Notes: `phMessage` is the handle of the JMS message being received. An XML document is created and 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

Procedure that handles any reply to a message that has been sent.

Location: `xml.p`

Parameters:

INPUT `phMessage` AS HANDLE

Handle of the JMS reply message being received.

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.

sendHandler

Procedure that sets the body of an outgoing message.

Location: `xml.p`

Parameters:

INPUT `phMessage` AS HANDLE

Handle of the JMS message being sent.

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.

sendReplyHandler

Procedure that processes a reply to a message.

Location: `xml.p`

Parameters:

INPUT `phMessage` AS HANDLE

Handle of the JMS reply message being sent.

Notes: An override of this procedure should set any necessary properties as well as supply the body of the reply message.

Methods for busines to business messaging Objects

The following sections lists and describes the methods for business to business Messaging objects.

callOutParams

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

Location: `b2b.p`

Parameters:

INPUT `pdNode` AS DECIMAL

Returns: LOGICAL

Notes: None

characterValue

Procedure that traps the event of a text node being parsed.

Location: b2b.p

Parameters:

INPUT pcPath AS CHARACTER

INPUT pcValue AS CHARACTER

Notes: None

createSchemaAttributes

Returns a new query handle for the schema temp-table to use for navigation through the children or a schema node.

Location: b2b.p

Parameters:

INPUT piParentNode AS INTEGER

Returns: HANDLE

Notes: At present, the caller is responsible for deleting the object.

createSchemaChildren

Returns a new query handle for the schema temp-table to use for navigation through the children of a schema node.

Location: b2b.p

Parameters:

INPUT piParentNode AS INTEGER

Returns: HANDLE

Notes: At present, the caller is responsible for deleting the object.

createSchemaPath

Returns a new query handle for the schema temp-table to use for navigation through the children of a schema node.

Location: b2b.p

Parameters:

INPUT pcPath AS CHARACTER

Returns: HANDLE

Notes: At present, the caller is responsible for deleting the object.

dataSource

Returns the handle of the data source identified by the argument.

Location: b2b.p

Parameters:

INPUT pcName AS CHARACTER

Returns: HANDLE

Notes: None

defineMapping

Creates a tMapping temp-table record.

Location: b2b.p

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.

destroyObject

Procedure that overrides in order to delete the schema temp-table.

Location: b2b.p

Parameters: None

Notes: None

endDocument

Procedure that traps the document-processed event and passes all the collected data that has been stored on other events during the processing.

Location: b2b.p

Parameters: None

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

Procedure that traps the XML-element-processed event. Stub routine meant to be overridden.

Location: b2b.p

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 pcQualifiedName AS CHARACTER

Name qualified with the namespace prefix.

Notes: This is just an empty stub to override. It really belongs in xml.p, or xml.p should use no-error.

findDataRow

Finds a row of a mapped DataObject. If any of the KeyFields are blank, use ForeignFields from the object's DataSource.

Location: b2b.p

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

initializeObject

Procedure that loads schema.

Location: b2b.p

Parameters: None

Notes: None

loadProducerSchema

Returns the result of identifying and loading the appropriate producer schema.

Location: b2b.p

Parameters: None

Returns: LOGICAL

Notes: None

loadSchema

Loads the schema temp-table.

Location: b2b.p

Parameters:

INPUT pcSchema AS CHARACTER

Returns: LOGICAL

Notes: None

mapNode

Maps a node to a procedure, function, or column in a data object.

Location: b2b.p

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().

NotFoundError

Returns a useful ‘Record not found’ error message with keyfields.

Location: b2b.p

Parameters:

INPUT phDataSource AS HANDLE

DataSource.

INPUT pcKeyValues AS CHARACTER

CHR(1) list with keyvalues corresponding to KeyFields.

Returns: CHARACTER

Notes: None

numParameters

Returns the number of parameters for a method.

Location: b2b.p

Parameters:

INPUT phProc AS HANDLE

Valid procedure handle.

INPUT pcMethod AS CHARACTER

Function or procedure name.

Returns: INTEGER

Notes: None

processMappings

Procedure that creates tMapping temp-table records from DirectionList, NameList, SchemaList, DocTypeList, DestinationList, ReplyReqList, and ReplySelectorList property values.

Location: b2b.p

Parameters: None

Notes: None

processMessages

Procedure that is called from receiveHandler() when an error occurs.

Location: b2b.p

Parameters:

INPUT pcMessages AS CHARACTER

Chr(3) and chr(4) delimited list, as returned from fetchMessages().

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.

produceAttributes

Procedure that produces attributes of schema nodes.

Location: b2b.p

Parameters:

INPUT piParentSchemaNode AS INTEGER

INPUT pdOwnerNode AS DECIMAL

INPUT phDataSource AS HANDLE

Notes: None

produceChildren

Procedure that produces all child nodes of an element or document.

Location: b2b.p

Parameters:

INPUT piParentSchemaNode AS INTEGER

Schema parent node.

INPUT pdParentNode AS DECIMAL

Actual parent node.

INPUT phDataSource AS HANDLE

LINKed data source of parent.

Notes: None

produceDocument

Procedure that creates a document and runs produceChildren() on it.

Location: b2b.p

Parameters: None

Notes: None

receiveHandler

Procedure that handles receiving a message.

Location: b2b.p

Parameters:

INPUT phMessage AS HANDLE

Handle of the JMS message being received.

Notes: The xm1.p SUPER creates an XML document and loads it from the message.

rowNotFoundError

Returns a “Record not found” error message with key fields.

Location: b2b.p

Parameters:

INPUT phDataSource AS HANDLE

 DataSource.

INPUT pcKeyValues AS CHARACTER

 CHR(1) list with KeyValues corresponding to KeyFields.

Returns: CHARACTER

Notes: None

schemaField

Returns the buffer-value of a schema field.

Location: b2b.p

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.

sendHandler

Procedure for event handler for sending the message.

Location: b2b.p

Parameters:

INPUT phMsgHandler AS HANDLE NO-UNDO.

Handle of message being sent.

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.

sendMessage

Procedure that overrides the msghandler message in order to set message-specific properties.

Location: b2b.p

Parameters: None

Notes: None

startDataRow

Registers the dataRow record used to find, create, or update the data.

Location: b2b.p

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

Procedure that traps the event of a new XML element being processed, checks whether it is mapped to a dataObject, and, if so, registers it.

Location: b2b.p

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 pcQualifiedName AS CHARACTER

Name qualified with the namespace prefix.

Notes: None

storeNodeValue

Store node values for a data source until the endDocument event. (Consume XML).

Location: b2b.p

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

Store mapping information for Output parameters (Produce XML).

Location: b2b.p

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

Stores values for input parameters.

Location: b2b.p

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.

targetProcedure

Procedure used by the router to subscribe to an event of the schema name in order to find whether it is already started.

Location: b2b.p

Parameters:

OUTPUT phHandle AS HANDLE.

phHandle = TARGET-PROCEDURE.

Notes: None

Methods for router messaging objects

The following section lists and describes the methods for router Messaging objects.

createDocument

Creates an empty document.

Location: router.p

Parameters: None

Returns: LOGICAL

Notes: Override of xml.p that does not delete the current document.

initializeObject

Procedure that initializes objects of class Router.

Location: router.p

Parameters: None

Notes: None

internalSchemaFile

Returns the internal Schema filename based on the external and internal file reference definitions.

Location: router.p

Parameters:

INPUT pcNameSpace AS CHAR

Target Namespace, the XMLNS attribute from the incoming XML document.

Returns: CHARACTER (filename)

Notes: None

obtainInMsgTarget

Procedure that 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.

Location: router.p

Parameters:

INPUT phMessage AS HANDLE

OUTPUT phInMessageTarget AS HANDLE

Notes: None

processFileRefs

Procedure that creates tFileReference temp-table records from ExternalRefList and InternalRefList property values.

Location: router.p

Parameters: None

Notes: None

routeBytesMessage

Procedure that takes a procedure that has a BytesMessage, loads the message, and sends the document to a SmartB2BObject.

Location: router.p

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.

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

Procedure that routes the documentHandle to a SmartB2BObject, including starting the SmartContainer with the SmartB2B if required, and giving it the loaded schema.

Location: router.p

Parameters:

OUTPUT pohInMessageTarget AS HANDLE

The SmartB2B that received the document.

Notes: None

routeMessage

Procedure that takes a procedure that has a message, loads the message, and sends the document to a B2B object and returns its handle.

Location: router.p

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.

Notes: Called directly from obtainInMesages or from specific route *MessageType* procedures.

startB2BObject

Starts the Container with the SmartB2BObject, links it to the consumer, and returns its handle.

Location: router.p

Parameters:

INPUT pcContainer AS CHARACTER

Returns: HANDLE of the container

Notes: None

Message object properties

Messaging Object properties provide information about messaging objects and their classes. This information can include whether an object is enabled, the contents of the object and so on. You can read property values and in many instances you can change property values. To read a value for a property, you use a **get** function and to change a value for a property, you use a **set** function.

These functions conform to the following conventions:

- **get** — Uses the form `get $\textit{propnam}$` and returns the current value of the property. This function accepts no arguments.
- **set** — Uses the form `set $\textit{propname}$` . The set function accepts a single argument—the new value for the property—and returns TRUE/FALSE depending on whether the value change succeeds.

For more information about getting and setting property values, see [Chapter 1, “ADM2 SmartObject API Reference”](#) in this guide.

ClientID

Property that stores the Client ID for the JMS broker connection. This property value is used during creation of the JMS session to set the ClientID of the JMS session. It should only be set before initialization of the messaging object.

Data Type: CHARACTER

Notes: Read and Write

ConsumerSchema

XML Schema filename.

Data Type: CHARACTER

Notes: Read only

ContextForServer

Value for `gcContextForServer`.

Data Type: LOGICAL

Notes: Write only

CurrentMessage

Property that stores the handle of the current message being produced. This property is set when the message is created by the producer's `sendMessage`, it should not be set in any other way. The handle of the message is sent to the send handler, therefore in most circumstances, it should not be necessary to get this handle for use in application code.

Data Type: HANDLE

Notes: Read and Write

CurrentMessageId

ID from the previous `sendMessage` where `ReplyRequired`.

Data Type: CHARACTER

Notes: Read and Write

Destination

Property that stores the destination of the message being sent. This would usually be set by the instance properties of the message handling object (`SmartB2BObject` or `SmartSender`) that is sending the message. It could be set after initialization of the object before the message is sent.

Data Type: CHARACTER

Notes: Read and Write

DestinationList

Value of `DestinationList`.

Data Type: CHARACTER

Notes: Read and Write

Destinations

Property that stores a CHR(1) delimited list of destinations from which a `SmartConsumer` object can receive messages. This property is set from the instance properties of the object and used during initialization of the object to create message consumers for each destination.

Data Type: CHARACTER

Notes: Read and Write

DirectionList

Value of DirectionList.

Data Type: CHARACTER

Notes: Read and Write

DocumentElement

ID of the root element.

Data Type: DECIMAL

Notes: Read only

DocumentHandle

Property that stores the XML document handle.

Data Type: HANDLE

Notes: Read and Write

DocumentInitialized

Determines if the document is initialized. For example, it has a root node.

Data Type: LOGICAL

Notes: Read only

Domain

Property that stores the messaging domain (Publish and Subscribe or Point-to-Point) for the SmartProducer and SmartConsumer messaging objects to determine how the JMS session is started for the object. This property can only be set before initialization of the messaging object and cannot be changed once the object is initialized.

Data Type: CHARACTER

Notes: Read and Write

DTDPublicId

DTDPublicId property of the Document.

Data Type: CHARACTER

Notes: Read and Write

DTDPublicIdList

CHR(1)- separated list of DTD Public Ids for producer.

Data Type: CHARACTER

Notes: Read and Write

DTDSysId

DTDSysId property of the Document.

Data Type: CHARACTER

Notes: Read and Write

DTDSysIdList

CHR(1)-separated list of DTD System Ids for producer.

Data Type: CHARACTER

Notes: Read and Write

ExternalRefList

List of external references this router uses to determine how external target namespaces map to internal XML mapping schemas.

Data Type: CHARACTER

Notes: Read and Write

InMessageSource

Handle that accesses the source of a message.

Data Type: HANDLE

Notes: Read and Write

InternalRefList

Comma-separated list. You can change this list by calling `modifyListAttribute`.

Data Type: CHARACTER

Notes: Read and Write

JMSpartition

Value of JMSpartition.

Data Type: CHARACTER

Notes: Read and Write

JMSPassword

Property that stores the password for the JMS broker connection for messaging objects (SmartProducer and SmartConsumer objects).

Data Type: CHARACTER

Notes:

- Read and Write
- This property must be set before initialization objects because it must be set in the JMS session before the JMS session begins.

JMSsession

Handle of the JMS session this instance is using.

Data Type: CHARACTER

Notes: Write only

JMSuser

Property that stores the user for the JMS broker connection for messaging objects (SmartProducer and SmartConsumer objects).

Data Type: CHARACTER

Notes:

- Read and Write
- This property must be set before initialization objects because it must be set in the JMS session before the JMS session begins.

LoadedByRouter

Determines whether or not the XML and Schema have already been loaded by the router. If TRUE, the XML and Schema have already been loaded.

Data Type: LOGICAL

Notes: Read and Write

LogFile

Property that stores the name of the log file written to for a SmartConsumer object running in batch. This should be set before initialization of the object

Data Type: CHARACTER

Notes: Read and Write

MapNameProducer

Value of MapNameProducer.

Data Type: CHARACTER

Notes: Read and Write

MapObjectProducer

Value of MapObjectProducer.

Data Type: CHARACTER

Notes: Read and Write

MapTypeProducer

Value of MapTypeProducer.

Data Type: CHARACTER

Notes: Read and Write

MessageType

Identifies the type of message being sent.

Data Type: CHARACTER

Notes: Read and Write

NameList

Value of NameList.

Data Type: CHARACTER

Notes: Read and Write

NamespaceHandle

Handle of the loaded XML mapping schema namespaces.

Data Type: HANDLE

Notes: Write only

NewNode

Identifies the next unused node number in sequence.

Data Type: DECIMAL

Notes:

- Read only
- Returns TRUE if the current RowObject record is in new mode. Returns "?" if there is no current RowObject.
- Returns TRUE if the matching contained SDO is in NewMode. This is the SBO version of getNewMode.

OutMessageSource

Property that stores the handle of the OUTMESSAGE-SOURCE object (SmartSender or SmartB2Bobject) of a SmartProducer object. This is set during creation of the SmartProducer object and should not be set by application code

Data Type: HANDLE

Notes: Read and Write

Persistency

Property that stores the persistency for messages sent in a SmartProducer's object's session. This property is initially set during initialization based on instance property values but can be set at any time to affect any messages sent after it is set.

Data Type: CHARACTER

Notes: Read and Write

PingInterval

Value of the ping interval for the session.

Data Type: INTEGER

Notes: Read and Write

Priority

Property that stores the priority for messages sent in a SmartProducer's object's session. This property is initially set during initialization based on instance property values but can be set at any time to affect any messages sent after it is set.

Data Type: INTEGER

Notes: Read and Write

PromptLogin

Determines whether or not the producer, consumer, or both prompts the user for the JMS broker login.

Data Type: LOGICAL

Notes: Read and Write

ReplyReqList

Value of ReplyReqList.

Data Type: CHARACTER

Notes: Read and Write

ReplyRequired

Property that stores a value that determines whether a reply is required for a message being sent for a message handling object (SmartB2BObject acting as a producer or SmartSender). This property is initially set during initialization based on instance property values but can be set at any time to affect messages sent after it is set.

Data Type: LOGICAL

Notes: Read and Write

ReplySelector

Property that stores a reply selector value to be used for messages that require replies that are being sent for a message handling object (SmartB2Bobject or SmartSender).

Data Type: CHARACTER

Notes: Read and Write

ReplySelectorList

Value of ReplySelectorList.

Data Type: CHARACTER

Notes: Read and Write

RouterSource

Property that stores the handle(s) of the ROUTER-SOURCE objects for a SmartRouter object.

Data Type: CHARACTER

Notes: Read and Write

SchemaHandle

Handle of the loaded XML mapping schema.

Data Type: HANDLE

Notes: Read and Write

SchemaList

List of the schema this B2B uses.

Data Type: CHARACTER

Notes: Read and Write

SchemaManager

Starts the schema manager, if necessary, and returns its procedure handle.

Data Type: HANDLE

Notes: Read only

Selectors

Represents the message selectors used when receiving messages.

Data Type: CHARACTER

Notes: Read and Write

ShutDownDest

Queue to which a message can be sent to shut down a SmartConsumer running unattended.

Data Type: CHARACTER

Notes: Read and Write

Subscriptions

Represents the information this consumer uses when subscribing to topics (only for Pub/Sub domain).

Data Type: CHARACTER

Notes: Read and Write

SupportedMessageTypes

Property that stores the Sonic Adapter supported message types. It is used by the SmartProducer's instance property dialog to provide a combo-box of supported messages types used to choose the message type to be stored in the MessageType property for that object.

Data Type: CHARACTER

Notes: Read only

TargetNameSpace

Defines the XMLNS attribute of the document instance.

Data Type: CHARACTER

Notes: Read only

TimeToLive

Represents the time during which the message is considered current and not stale.

Data Type: DECIMAL

Notes: Read and Write

TypeName

Identifies the documentation, destination, or both for multi-document producers.

Data Type: CHARACTER

Notes: Read and Write

UseDTD

Determines whether or not to use DTD. TRUE, if DTD is to be used.

Data Type: LOGICAL

Notes: Read only

ValidateOnLoad

Determines whether or not the document should be validated on load.

Data Type: LOGICAL

Notes: Read and Write

Waiting

Determines how long the adapter waits for a message.

Data Type: LOGICAL

Notes: Read and Write

Alphabetical Listing Of WebSpeed-specific API Routines

The WebSpeed environment coexists with and makes use of HTML. In this respect, it is different than the standard 4GL/ADM environment, and SmartObjects operating in it require a separate set of supporting routines.

This section lists all SmartObject API routines dedicated to the WebSpeed environment. For ADM2 routines not specific to the WebSpeed environment, see the other chapters in this guide. For lists of methods, procedures, and properties organized by object type, see the Index.

The default paths to the files defining the SmartObjects for WebSpeed are:

- **Class SourcePath** — `src/web2`
- **Class RcodePath** — `web2`
- **Class TemplatePath** — `src/web2/template`

addColumnLink

Specifies that a column in the HTML table should appear with a hyperlink. The attributes needed to support this are specified in the other input parameters.

Location: `wbtable.p`

Parameters:

INPUT *pcColumn* AS CHARACTER

The name of the column you want hyperlinked.

INPUT *pcURL* AS CHARACTER

The linked object. Must be a valid URL reference.

INPUT *pcTarget* AS CHARACTER

The HTML frame reference that receives the response. The reference `_self` specifies that the current HTML frame should be used.

INPUT *pcMouseOver* AS CHARACTER

A function that returns the character string that should be displayed on the mouseOver event in the HTML page.

INPUT *pcJoinParam* AS CHARACTER

Specifies the required parameters to add to the URL 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

Adds fields to the list of fields that are used to keep context for the next request.

Location: webrep.p

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

Adds SearchName and SearchValue to the data-source query.

Location: webrep.p

Parameters:

INPUT pcColumn AS CHARACTER

The Column's name in the data-source.

INPUT pcValue AS CHARACTER

Search value.

Returns: LOGICAL

Notes: None

addTDModifier

Adds one or more HTML attributes for the <TD> tag of a specific column.

Location: wtable.p

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

Determines whether there are any messages in the message queue.

Location: webrep.p

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.

AddMode

ADDMode property. If set to TRUE, the assignFields procedure creates a new record.

Notes: Write only

assignColumnFormat

Sets a column's format.

Location: webrep.p

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

Overrides default HELP.

Location: webrep.p

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

Overrides the default label.

Location: webrep.p

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

Unused function.

Location: webrep.p

Parameters:

INPUT pColumn AS CHARACTER

INPUT pwidth AS DECIMAL

Returns: LOGICAL

Notes: Currently not in use.

assignExtentAttribute

Enables the user to override field attributes for columns with extents.

Location: webrep.p

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

Procedure that assigns the current form buffer values in the enabled fields/objects to the SmartDataObject, the database, and/or to local object variables.

Location: html-map.p

Parameters: None

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

Procedure that 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.

Location: wbdata.p

Parameters: None

Notes:

- The DataColumn 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

Assigns one or more HTML attributes for the <TD> tag of a specific column.

Location: whtable.p

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 */  
assignTDM Modifier("custnum", 'align="left" bgcolor = "#FFFa00"').
```

Attribute

The name of the Web-related 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.
```

Notes: Read only

bufferHandle

Gets the handle of a buffer by name.

Location: html-map.p

Parameters:

INPUT pcTableName AS CHARACTER

The name of a table in the query or object.

Returns: HANDLE

Notes: None

bufferHandle

Gets the handle of a buffer by table name.

Location: webrep.p

Parameters:

INPUT pcTableName AS CHARACTER

The name of a table in the query.

Returns: HANDLE

Notes: None

columnDataType

Returns a column's datatype from the data-source.

Location: webrep.p

Parameters:

INPUT pcColumn AS CHARACTER

The Column's name in the data-source.

Returns: CHARACTER

Notes: None

columnFormat

Returns a columns format from the data-source.

Location: webrep.p

Parameters:

INPUT pcColumn AS CHARACTER

The column's name in the data-source.

Returns: CHARACTER

Notes: None

columnHandle

Returns the buffer handle of a column in the internal query.

Location: webrep.p

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

Return a column's help text from the data-source.

Location: webrep.p

Parameters:

INPUT pcColumn AS CHARACTER

The column's name in the data-source.

Returns: CHARACTER

Notes: None

columnHTMLName

Finds the HTML field name of a column.

Location: html-map.p

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

Returns a unique valid HTML identifier/name for the column.

Location: webrep.p

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

Returns the Column's label.

Location: webrep.p

Parameters:

INPUT pcColumn AS CHARACTER

The column's name in the data-source.

Returns: CHARACTER

Notes: None

columnReadOnly

Returns TRUE if the column is non updatable.

Location: webrep.p

Parameters:

INPUT pcColumn AS CHARACTER

The column's name in the data-source.

Returns: LOGICAL

Notes: None

columnStringValue

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**.

Location: webrep.p

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 [getUpdateMode](#).

columnTable

Overrides to handle HTML mapping objects with neither query nor SDO.

Location: `html-map.p`

Parameters:

INPUT *pcColumn* AS CHARACTER

Returns: CHARACTER

Notes: None

columnTDModifier

Returns the HTML attributes for the <TD> tag of a specific column.

Location: `wbtable.p`

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

Returns the validation message of a particular column. This is the text that serves as the error message when a validation fails.

Location: wldata.p

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

Procedure that 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.

Location: html-map.p

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.

Notes: Related routine: adm-create-objects

dataAvailable

Procedure that overrides dataAvailable.

Location: `html-map.p`

Parameters:

INPUT `pcMode` AS CHARACTER

See `query.p`

Notes: This is a workaround to avoid running `query.p` logic when this object uses a data-source.

deleteBuffer

Deletes the current record of a query's buffer from the database. Returns TRUE when it succeeds and FALSE if it fails.

Location: `wbdata.p`

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

Procedure that deletes all HTML field offset temp-table records for a HTML mapping procedure.

Location: `html-map.p`

Parameters: None

Notes: Intended for internal use only.

deleteRow

Delete the current row from the data-source.

Location: `wbdata.p`

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

Procedure that deletes this Web object's associated offset records in memory and this object if it was run PERSISTENT.

Location: `admweb.p`

Parameters: None

Notes: None

destroyDataObject

Shuts down the current data object.

Location: webrep.p

Parameters: None

Returns: LOGICAL

Notes: None

destroyObject

Procedure that destroys the procedure including the offset.

Location: admweb.p

Parameters: None

Notes: None

destroyObject

Procedure that performs any necessary cleanup before the object is destroyed.

Location: webrep.p

Parameters: None

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

Procedure that disconnects from the AppServer if there is one.

Location: webrep.p

Parameters: None

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

Procedure that calls the standard utility procedure, as defined in `tagmap.dat`, for the current Web object.

Location: `html-map.p`

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

Notes: Intended for internal use only.

displayFields

Procedure that copies SmartDataObject or database values to the displayed object and field form buffer screen values.

Location: `html-map.p`

Parameters: None

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

Procedure that enables object and field form buffer widgets by setting their SENSITIVE attribute to TRUE.

Location: html-map.p

Parameters: None

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

Exclusively locks a buffer in the query.

Location: wldata.p

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

Returns the attribute for a variable with extent.

Location: webrep.p

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

Returns the extent number if a field name has brackets.

Location: webrep.p

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

Procedure that repositions the data-source query to the rowids stored in the CurrentRowids property.

Location: webrep.p

Parameters: None

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

Procedure that repositions the data-source query to the first row.

Location: webrep.p

Parameters: None

Notes: None

fetchLast

Procedure that repositions the query of the data-source to the first row of the last page

Location: wtable.p

Parameters: None

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

Procedure that repositions the data-source to the query's last row.

Location: webrep.p

Parameters: None

Notes: None

fetchNext

Procedure that repositions the query of the data-source to the first row of the next page.

Location: wtable.p

Parameters: None

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

Procedure that repositions the data-source query to the next row.

Location: `webrep.p`

Parameters: None

Notes: The procedure re-establishes context with a call to `fetchCurrent` before the actual navigation takes place.

fetchPrev

Procedure that repositions the query of the data-source to the first row of the previous page.

Location: `whtable.p`

Parameters: None

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

Procedure that repositions the data-source query to the previous row.

Location: `webrep.p`

Parameters: None

Notes: The procedure re-establishes context with a call to `fetchCurrent` before the actual navigation takes place.

fieldExpression

Creates an expression for a field.

Location: webrep.p

Parameters:

INPUT pColumn AS CHARACTER

INPUT pOperator AS CHARACTER

INPUT pValue AS CHARACTER

Returns: CHARACTER

Notes: None

findRecords

Procedure that opens the SmartDataObject or database query that finds the records in this frame.

Location: html-map.p

Parameters: None

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.
```

getAttribute

Returns the value of a standard Web-related attribute.

Location: admweb.p

Parameters:

INPUT p_attr-name AS CHARACTER

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.

Returns: CHARACTER (value of the attribute)

Notes: None

Examples:

```
RUN getAttribute ("web-state").
cWebState = RETURN-VALUE.
```

getContextFields

Retrieves the ContextFields property (see webprop.i), a comma-separated list of Fields for which URL parameters are needed to keep context for subsequent requests to this object.

Location: webrep.p

Parameters: None

Returns: CHARACTER

Notes:

- 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.

getCurrentPage

Called by adm-create-objects. Checks the current page.

Location: `html-map.p`

Parameters: None

Returns: INTEGER

Notes: Intended for internal use only.

getCurrentRowids

Returns the String of comma-delimited rowids in the CurrentRowid property. (See `webprop.i.`)

Location: `webrep.p`

Parameters: None

Returns: CHARACTER

Notes:

- 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.

getDeleteTables

Retrieves a comma-separated list of tables to be deleted.

Location: `wbdata.p`

Parameters: None

Returns: CHARACTER

Notes: The `getTables` is used if this attribute is not set.

getForeignFieldList

Retrieves the alternative list of foreign fields corresponding to the ExternalTableList.

Location: webrep.p

Parameters: None

Returns: CHARACTER

Notes: None

getFrameHandle

Returns the frame handle of the object.

Location: wbdata.p

Parameters: None

Returns: HANDLE

Notes: None

getNavigationMode

Returns the current navigation mode. NOT IN USE.

Location: wbrep.p

Parameters: None

Returns: CHARACTER

Notes: Not in use.

getNextHtmlField

Procedure that reads the HTML file one line at a time, sending each line to the Web stream, up to the next HTML field definition.

Location: `html-map.p`

Parameters:

INPUT-OUTPUT `next-line` AS CHARACTER

Full text of current line.

INPUT-OUTPUT `line-no` AS INTEGER

Line position counter.

INPUT-OUTPUT `start-line-no` AS INTEGER

Beginning field definition line.

INPUT-OUTPUT `start-line-offset` AS INTEGER

Beginning field definition column.

INPUT-OUTPUT `end-line-no` AS INTEGER

Ending field definition line.

INPUT-OUTPUT `end-line-offset` AS INTEGER)

Ending field definition column.

INPUT-OUTPUT `field-def` AS CHARACTER

Field definition between `<` and `>`

INPUT-OUTPUT `clip-bytes` AS INTEGER)

Chars on line already processed.

Notes: Internal use only. The field definition is extracted from the [row, column] offsets [start-line-no, start-line-offset] to [end-line-no, end-line-offset] and returned.

getQueryEmpty

Checks to verify whether or not the data source or query is empty.

Location: webrep.p

Parameters: None

Returns: LOGICAL

Notes: None

getQueryWhere

Gets the current WHERE clause for the query.

Location: wbrep.p

Parameters: None

Returns: CHARACTER

Notes: None

getRowids

Gets the ROWID(s) of the current row or rows.

Location: wbrep.p

Parameters: None

Returns: CHARACTER

Notes: None

getSearchColumns

Gets the SearchColumns property. (see webprop.i.)

Location: webrep.p

Parameters: None

Returns: CHARACTER

Notes: Contains a comma-delimited list of columns that are used to search in the current data-source. Currently supports only one column.

getServerConnection

Returns SESSION:SERVER-CONNECTION-ID through ServerConnection property.

Location: wbrep.p
Parameters: None
Returns: CHARACTER
Notes: None

getTableRowids

Retrieves a list of current rowids by using getRowids, and then uses the ExternalTables property (see webprop.i) to remove the External Tables from the list, returning a list of rowids for only the records requested by the current object.

Location: webrep.p
Parameters: None
Returns: CHARACTER
Notes: Used to pass as external rowids to a called object.

getTableRows

Returns the number of rows in the table.

Location: wbtable.p
Parameters: None
Returns: INTEGER
Notes: None

getTables

Overrides to use the property for HTML mapping with no data-source.

Location: html-map.p
Parameters: None
Returns: CHARACTER
Notes: None

getTables

Returns a comma-delimited list of tables from which data will be retrieved, either from a SmartDataObject or database.

Location: webrep.p

Parameters: None

Returns: CHARACTER

Notes: None

getUpdateMode

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.

Location: webrep.p

Parameters: None

Returns: CHARACTER

Notes: None

getWebState

Returns the Web state for a Web object.

Location: admweb.p

Parameters: None

Returns: CHARACTER (Possible values are state-aware, state-less, timed-out.)

Notes: None

Examples:

```
cWebState = getWebState().
```

getWebTimeout

Returns the timeout in minutes for a state-aware Web object.

Location: admweb.p

Parameters: None

Returns: DECIMAL (Timeout in minutes)

Notes: None

Examples:

```
dWebTimeout = getWebTimeout().
```

getWebTimeRemaining

Returns the time remaining for a state-aware Web object.

Location: admweb.p

Parameters: None

Returns: DECIMAL (Time remaining)

Notes: None

Examples:

```
dWebTimeout = getWebTimeRemaining().
```

getWebToHdlr

Returns the name of the Web object or procedure to run when the currently running state-aware Web object times out.

Location: admweb.p

Parameters: None

Returns: CHARACTER (Name of the object/procedure to run.)

Notes: None

Examples:

```
cWebToHdlr = getWebToHdlr().
```

htmAssociate

Procedure that maps HTML fields to their Web object widget counterparts.

Location: `html-map.p`

Parameters:

INPUT `htmField` AS CHARACTER

HTML field name.

INPUT `wdtField` AS CHARACTER

Web object field name.

INPUT `widHandle` AS HANDLE

Web object field handle.

Notes: Internal use only. AppBuilder-maintained, read-only. The essential HTML mapping procedure.

HTMLAlert

Generates an alert-box in the HTML.

Location: `webrep.p`

Parameters:

INPUT `pcMessage` AS CHARACTER

The message to display.

Returns: LOGICAL

Notes: The message is generated with JavaScript.

HTMLColumn

Outputs the columns value within an HTML <TD> tag, including the HREF definitions to create defined hyperlinks.

Location: `wbtable.p`

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

Sets focus to a field in a WebPage.

Location: `webrep.p`

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

Outputs the entire HTML <TABLE>.

Location: whtable.p

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

Procedure that sets the list of displayed and enabled objects, fields, or both.

Location: html-map.p

Parameters: None

Notes: Intended for internal use only.

inputFields

Procedure that receives field input from the Web browser and populates the form buffer field values.

Location: html-map.p

Parameters: None

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

Adds external tables to the buffers in the database query.

Location: webrep.p

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

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.

Location: webrep.p

Parameters:

INPUT pcObject AS CHARACTER

A table or SmartDataObject name that has an entry in the ForeignFieldsList.

Returns: LOGICAL

Notes: None

lockRow

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.

Location: wbdata.p

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 returns FALSE and adds 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

Opens the database query in the data-source.

Location: webrep.p

Parameters: None

Returns: LOGICAL

Notes: None

outputFields

Procedure that replaces the tagged HTML field definition with the data values stored in the form buffer.

Location: html-map.p

Parameters: None

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

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.

Location: whtable.p

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

Procedure that 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.

Location: wbdata.p

Parameters: None

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 mode of an object 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

Procedure that 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.

Location: `wbtable.p`

Parameters: None

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

Procedure that processes the submitted request from the Web.

Location: `webrep.p`

Parameters: None

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

Procedure wrapper that wraps a procedure that reads the HTML mapping offset file and populates internal AppBuilder temp-tables.

Location: `html-map.p`

Parameters:

INPUT `cWebFile` AS CHARACTER

Name of offset file.

Notes: Internal use only.

removeEntry

Removes an entry from a list.

Location: `webrep.p`

Parameters:

INPUT `pNum` AS INTEGER

INPUT `pList` AS CHARACTER

Returns: CHARACTER PRIVATE

Notes:

- PRIVATE, not recommended for use.
- Delimiter is always comma.

reOpenQuery

Resets the data-source query to its original state.

Location: webrep.p

Parameters: None

Returns: LOGICAL

Notes: This is the state after context has been reset, but before a search or query has been performed.

rowidExpression

Creates ROWID expression for dynamic query Rowid(table) = to-rowid(rowidchar).

Location: webrep.p

Parameters:

INPUT pcBuffer AS CHARACTER

Buffer name in the query.

INPUT pcRowid AS CHARACTER

STRING of the Rowid.

Returns: CHARACTER PRIVATE

Notes: None

setAddMode

Sets the ADDMode property. If set to TRUE, the assignFields procedure will create a new record.

Location: wbdata.p

Parameters:

INPUT p1Add AS LOGICAL

The new value for the property.

Returns: LOGICAL

Notes: None

setAppService

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 if it is connected to a different partition.

Location: wbre.p

Parameters:

INPUT pAppService AS CHARACTER

The AppService in which the SDO will start.

Returns: LOGICAL

Notes: This value will be passed to the SDO before initialization.

set-attribute-list

Accepts the value of the complete object attribute list and runs procedures to set individual attributes.

Location: admweb.p

Parameters:

INPUT p-attr-list AS CHARACTER

A comma-separated attribute list with the format **name=value**. Typical attributes are web-timeout, web-state, and web-timeout-handler. In addition, other names can be used provided that the special-get-attribute procedure to handle them exists in the target procedure.

Returns: None

Notes: Not all attributes can be set. Those that are a part of an event such as enable/disable (which set ENABLED on/off) or hide/view (which set HIDDEN on/off), can be queried through getAttribute, but are read-only.

Examples:

```
RUN set-attribute-list ("web-state=persistent, web-timeout=60").
```

setBuffers

Sets the `Buffers` property (see `webprop.i`), a comma-separated list of database tables (not `SmartDataObject`) to be retrieved by this object.

Location: `webrep.p`

Parameters:

INPUT `pcTables` AS CHARACTER

Tables to use in the database query.

Returns: LOGICAL

Notes: This will clear all previously defined buffers.

setColumns

Sets the `DataColumns` property (see `webprop.i`), a comma-separated list of database tables (not `SmartDataObject`) to be retrieved by this object.

Location: `webrep.p`

Parameters:

INPUT `pcColumns` AS CHARACTER

A comma-separated list of column names.

Returns: LOGICAL

Notes: The function is named `setColumns` instead of `setDataColumns` for backwards compatibility.

setContextFields

Sets the `ContextFields` property (see `webprop.i`), a comma-separated list of fields for which URL parameters are needed to keep context for subsequent requests to this object.

Location: `webrep.p`

Parameters:

INPUT `pcContextfield` AS CHARACTER

The new property.

Returns: LOGICAL

Notes: Use addContextFields to add to the list.

setCurrentRowids

Sets the String of comma-delimited rowids in the CurrentRowid property that will be used by fetchCurrent. (See webprop.i.)

Location: webrep.p

Parameters:

INPUT pcRowids AS CHARACTER

A list of ROWIDS.

Returns: LOGICAL

Notes:

- 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.

setDeleteTables

Stores a comma-separated list of tables to be deleted.

Location: wbdata.p

Parameters:

INPUT pcDeleteTables AS CHARACTER

Returns: LOGICAL

Notes:

- 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.

setExternalJoinList

Sets the ExternalJoinList property (see webprop.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.

Location: webrep.p

Parameters:

INPUT pcExternalJoinList AS CHARACTER

A pipe (|)-separated list of OF phrases.

Returns: LOGICAL

Notes: None

Examples:

```
/*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').
```

setExternalTableList

Sets the ExternalTableList property (see webprop.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.

Location: webrep.p

Parameters:

INPUT pcExternalTableList AS CHARACTER

A pipe (|)-separated list of comma-separated tables.

Returns: LOGICAL

Notes: The ExternalJoinList and ExternalWhereList and/or ForeignFieldList have corresponding entries.

Examples:

```

/* 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|')

```

setExternalTables

Sets the ExternalTables property (see webprop.i), a comma-separated list of the current List of ExternalTables to use to retrieve data for this request.

Location: webrep.p

Parameters:

INPUT pcExternalTables AS CHARACTER

A comma-separated list of table names.

Returns: LOGICAL

Notes: None

setExternalWhereList

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.

Location: `webrep.p`

Parameters:

INPUT `pcExternalWhereList` AS CHARACTER

A pipe (|)-separated list of field expressions.

Returns: LOGICAL

Notes:

- 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
```

setForeignFieldList

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.

Location: `webrep.p`

Parameters:

INPUT `pcForeignFieldList` AS CHARACTER

Returns: LOGICAL

Notes: None

setFrameHandle

Stores the handle of the frame.

Location: wldata.p

Parameters:

INPUT pHd1 AS HANDLE

Returns: LOGICAL

Notes: None

setLinkColumns

Stores the columns that have hyperlinks as a comma-separated list in the LinkURLs property (see wbtaprop.i).

Location: whtable.p

Parameters:

INPUT pLinkColumns AS CHARACTER

The comma-separated list of columns.

Returns: LOGICAL

Notes: None

Examples:

```
setLinkColumns ("custnum,salesrep")
```

setLinkColumns

Stores a comma-separated list of columns that have hyperlinks.

Location: webrep.p

Parameters:

INPUT pcLinkColumns AS CHARACTER

A comma-separated list of column names.

Returns: LOGICAL

Notes: None

setQueryWhere

Prepares the query with a new OPEN QUERY statement or a new expression.

Location: webrep.p

Parameters:

INPUT pcMode AS CHARACTER

The new WHERE clause or expression.

Returns: LOGICAL

Notes: None

setSearchColumns

Stores the SearchColumns value. (Currently one.)

Location: webrep.p

Parameters:

INPUT pcSearchColumns AS CHARACTER

A column name in the data-source.

Returns: LOGICAL

Notes: None

setServerConnection

Sets SERVER_CONNECTION_ID property from SESSION:SERVER-CONNECTION-ID.

Location: webrep.p

Parameters: None

Returns: LOGICAL

Notes: None

setTableModifier

Stores the Specified HTML attributes for the <table> tag in TableModifier property (see wbtaprop.i).

Location: whtable.p

Parameters:

INPUT pTableModifier AS CHARACTER

One or more HTML attributes that will be used in the TABLE tag.

Returns: LOGICAL

Notes: The Embedded SpeedScript (.htm) files created by the Report wizard show examples of this function call in use.

Examples:

```
setTableModifier(' border="2" cellspacing=10')
```

setTableRows

Stores the specified number of rows for the HTML table in TableRows property (see wbtaprop.i).

Location: whtable.p

Parameters:

INPUT PRows AS INTEGER

Number of rows to display in the HTML page.

Returns: INTEGER

Notes: The Embedded SpeedScript (.htm) files created by the Report wizard show examples of this function call in use.

Examples:

```
setTableRows(10)
```

setUpdateMode

Sets 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.

Location: webrep.p

Parameters:

INPUT pcMode AS CHARACTER

Either **yes** or **no**.

Returns: LOGICAL

Notes: 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.

setUseColumnLabels

Stores a logical specifying whether to use column labels for the HTML table in UseColumnLabels property (see wbtaprop.i).

Location: whtable.p

Parameters:

INPUT pUseLabels AS LOGICAL

Returns: LOGICAL

Notes: 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).

Examples:

setUseColumnLabels(no)

setWebState

Sets the Web state to state-aware and the timeout for Web objects.

Location: admweb.p

Parameters:

INPUT pdWebTimeout AS DECIMAL

The number of minutes to remain state-aware.

Returns: LOGICAL

Notes: None

Examples:

```
lReturn = setWebState(5.0).
```

setWebToHdlr

Sets the name of the Web object or procedure to run when the currently running state-aware Web object times out.

Location: admweb.p

Parameters:

INPUT pcWebToHdlr CHARACTER

The Web-object name.

Returns: LOGICAL

Notes: Web object must be on the Agent's PROPATH.

Examples:

```
lReturn = setWebToHdlr("mytohdlr.w").
```

showDataMessages

Runs `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.

Location: `webrep.p`

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

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.

Location: `webrep.p`

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

Procedure that sets the Web state to timed-out for a Web object.

Location: `admweb.p`

Parameters: None

Notes: RUN `timingOut` is equivalent to `setWebState (0)`.

urlJoinParams

Generate the URL parameter to use as join information for a linked object.

Location: webrep.p

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

Returns the necessary URL parameters to pass record information to a linked object.

Location: webrep.p

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

Returns TRUE if all of the database columns in the optionally defined frame are validated. The frame is generated specifically for this purpose.

Location: wbdata.p

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

Verifies whether or not a value is the correct data type.

Location: webrep.p

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.

Progress Dynamics Call Wrapper

The Progress Dynamics™ Call Wrapper provides an efficient way to dynamically invoke code with parameter lists that are defined at run time. The wrapper supports all required calls and combinations of parameters so that you can make calls to your business logic. This wrapper operates both inside and outside the existing Progress Dynamics framework so that you can use the functionality when ADM2 is running in a non-Dynamics environment.

If your application is a Web-based application, the call wrapper can obtain values to parameters from context that has been stored in the session. This context can be stored using either the `setPropertyList` or `setSessionParam`. The wrapper can also derive the parameters for a call from this context and can also write the return values from a call into this context. The wrapper can invoke procedures within the current session as well as on an AppServer.

You can invoke the call wrapper in the following ways: using the `dynlaunch.i` include file, using a single-entry-point procedure, or by using the API. This chapter provides information about:

- [Invoking the dynamics call wrapper using `dynlaunch.i`](#)
- [Invoking the dynamics call wrapper using a single-entry point](#)
- [Temp-table include files](#)
- [Temp-table types](#)
- [API Reference](#)
- [Invoking the dynamics call wrapper at the API level](#)

Invoking the dynamics call wrapper using dynlaunch.i

The `dynlaunch.i` include file provides the simplest and easiest way to invoke the Progress Dynamics™ Call Wrapper. Progress Software Corporation recommends that you use `dynlaunch.i` when using an AppServer where the procedure handle is not needed after the call.

`dynlaunch.i` accepts a temp-table of call information, constructs the call using this information and then invokes the call. Although using `dynlaunch.i` is the simplest way to invoke the call wrapper, the performance overhead is greater than when invoking the call wrapper using the single-entry point or at the API level. Therefore, Progress Software Corporation recommends that you use `dynlaunch.i` as follows:

- When making calls across the AppServer and the benefits of reduced AppServer calls and an unbound connection outweigh the disadvantage of the overhead associated with the dynamic call.
- When call parameters are only known at run time.

`dynlaunch.i` supports calls to an internal procedure inside a server-side procedure that might or might not already be running. When you invoke the dynamics call wrapper using `dynlaunch.i`, it handles all of the following in a single AppServer call:

- Identifies whether the external procedure is already running on the server and starts the procedure if it is not running.
- Runs the internal procedure inside the server-side persistent procedure.
- Gets back the OUTPUT parameters from the internal procedure call.
- Deletes the server-side procedure if it was started just for this call.

The following list the include file arguments for `dynlaunch.i`:

- **&PLIP** — Named argument that names the external procedure needed on the server. Alternatively, it can be the logical name (the Manager Type name) of any registered Progress Dynamics™ Manager, including a newly created manager.
- **&Iproc** — Named argument that names the internal procedure to run.
- **&clearHandlen** — Deletes the TABLE-HANDLE or BUFFER after the call completes.
- **&Define-only** — If set to Yes, you can define the variables that you need that require no action.

There must also be three named arguments for each parameter in the internal procedure's calling sequence. For each argument, the n represents the order of the parameter in the call:

- **&moden** — Named argument is INPUT, OUTPUT, or INPUT-OUTPUT.
- **&parmn** — Named argument that names the variable or table field storing the parameter.
- **&datatypen** — Named argument that holds the data type of the parameter.

For more information about using `dynlaunch.i`, see the [Progress Dynamics Developer's Guide](#) and the [Progress Dynamics Programming Handbook](#).

Invoking the dynamics call wrapper using a single-entry point

Using a single-entry point to invoke the dynamics call wrapper requires more coding compared to using `dynlaunch.i`. However, using this method to invoke the dynamics call wrapper provides greater control of error handling and improved performance.

When you use a single-entry point, you can invoke the dynamics call wrapper using any of the single-entry-point procedures to make the appropriate call using a single RUN statement. These single-entry-point procedures act as containers that perform the tasks needed to make a dynamic call.

NOTE: Progress Software Corporation recommends that you use dynamic calls only in situations where signatures are unknown at run time and the number of calls is relatively small.

The following sections describe these single-entry point procedures:

- [callstring.p](#)
- [callstringtt.p](#)
- [calltable.p](#)
- [calltablett.p](#)

callstring.p

The `callstring.p` procedure allows you to make a dynamics call by providing a minimal list of information. The procedure then evaluates the data needed to make the call and performs all the steps necessary to invoke the call. This is advantageous because `callstring.p` can be run from a client procedure in a single `AppServer` request.

The caller can invoke procedures and functions in a remote manager by providing mnemonics that are evaluated by the `callstring.p` on the server. These mnemonics map to handles to procedures that are not known to the client.

The procedure invokes calls using a string containing the parameters in the form:

mode datatype parameter, mode datatype parameter,...

Each token in a group is separated by a space, and when the value of a parameter token is a constant, the value is enclosed in single quotes.

NOTE: The following examples illustrate code for use in a dynamics environment because they use the Connection Manager.

This code example illustrates a valid `callstring.p` invocation:

```
RUN callstring.p
("disconnectService",          /* call to make */
 "ConnectionManager",         /* manager to use */
 "INPUT CHARACTER 'sports2000'"). /* parameter string */
```

The call results in a call to the `disconnectService` procedure in the Connection Manager in the local session and passes a single input parameter of data type character with the constant value of **sports2000**.

If the value of the parameter is determined from context, the constant value **sports2000** can be replaced with a context variable name:

```
RUN callstring.p
("disconnectService",          /* call to make */
 "ConnectionManager",         /* manager to use */
 "INPUT CHARACTER cDBName").  /* parameter string */
```

In this sample, **cDBName** is the name of a property or parameter that was previously set using `setPropertyList` or `setSessionParam`.

callstringtt.p

The `callstringtt.p` procedure is similar to `callstring.p`, except that it allows you to pass up to 64 temp-tables that function as parameters to the `invokeCall` procedure.

In addition to the behavior provided by `callstring.p`, `callstringtt.p` also allows the caller to provide a mapping between the 64 temp-tables being passed and the appropriate parameter to the call.

The code in [Example A-1](#) illustrates an example procedure that you might need to call in a server-side procedure.

Example A-1: Example server-side procedure

```
/* customerbl.p */
...

PROCEDURE obtainCustomerData:
    DEFINE INPUT PARAMETER phCustomerTempTable AS HANDLE.
    DEFINE INPUT PARAMETER pcSessionID AS CHARACTER.
    DEFINE INPUT PARAMETER phOrderBuffer AS HANDLE.

    /* Process the customer temp-table and order buffer */

END.
```

The code in [Example A–2](#) illustrates making a call to the `obtainCustomerData` procedure. In this example, the procedure passes the handle to the temp-table as the first parameter and the buffer handle for the order temp-table as the second parameter.

Example A–2: Making a call to the `obtainCustomerData` procedure

```
RUN adm2/callstringtt.p
("obtainCustomerData", /* call to make */
 "customerbl.p",      /* procedure to use */
 "INPUT HANDLE 'T:01', INPUT CHARACTER cSessionID, INPUT HANDLE
'B:02'",              /* parameter list */
 "",                  /* Tables to skip */
 INPUT-OUTPUT TABLE ttCust,
 INPUT-OUTPUT TABLE ttOrder,
 INPUT-OUTPUT TABLE-HANDLE htt03,
 INPUT-OUTPUT TABLE-HANDLE htt04,
 INPUT-OUTPUT TABLE-HANDLE htt05,
 INPUT-OUTPUT TABLE-HANDLE htt06,
 INPUT-OUTPUT TABLE-HANDLE htt07,
 INPUT-OUTPUT TABLE-HANDLE htt08,
 INPUT-OUTPUT TABLE-HANDLE htt09,
 INPUT-OUTPUT TABLE-HANDLE htt10,
 ...
 INPUT-OUTPUT TABLE-HANDLE htt64
 ).
```

NOTE: [Example A–2](#) illustrates code for use in a non-dynamics environment because it uses a procedure file.

You must specify each TABLE-HANDLE you want to pass in a remote call. As a result, to make a remote call to the AppServer, you must specify all 64 TABLE-HANDLES. To do this efficiently, you can specify the TABLE-HANDLE parameters using the include file `callttparam.i` located in the `src/adm2/` directory. Before using this include file, you must define an array of handles with an extent of 64.

To do this, you could modify the code in [Example A–2](#) as follows:

```

DEFINE VARIABLE hTT AS HANDLE EXTENT 64 NO-UNDO.

RUN adm2/callstringtt.p
  ("obtainCustomerData", /* call to make */
   "customerbl.p",       /* procedure to use */
   "INPUT HANDLE 'T:01', INPUT CHARACTER cSessionID, INPUT HANDLE
'B:02'",                 /* parameter list */
   "",                   /* Tables to skip */
   {src/adm2/callttparam.i
    &ARRAYFIELD = "hTT"
    &T01 = "TABLE ttCust"
    &T02 = "TABLE ttOrder"}
  ).

```

The syntax highlighted in bold in the [Example A–2](#) modified code deserves some additional explanation:

```

"INPUT HANDLE 'T:01', INPUT CHARACTER cSessionID, INPUT HANDLE 'B:02'"

```

The first and third parameters in the string correspond to the first and second temp-tables being passed as indicated by the 01 and 02 values after the colon. Note that the **T:** or **B:** that precedes the number indicates whether to pass a TEMP-TABLE (T) or BUFFER (B) handle to the procedure being called. As a result, the passing order of the temp-tables in the include file does not need to be the same order used by the call. The order used in the call is determined by the subscript that follows the **T:** or **B:**.

For more information, see the [“Temp-table include files”](#) and [“Temp-table types”](#) sections.

calltable.p

The `calltable.p` procedure allows invocation of a call by passing in a temp-table that contains the parameter values for each parameter. The temp-table must be in one of the forms supported in the `calltables.i` file located in the `src/adm2/` directory, or from the signature of the procedure or function being invoked.

For more information, see the [“Temp-table include files”](#) and [“Temp-table types”](#) sections.

The code in [Example A-3](#) illustrates using `calltable.p` to invoke a call.

Example A-3: Using `calltable.p` to invoke a call

```
DEFINE VARIABLE htt AS HANDLE NO-UNDO.

/* Include the temp-table definition */
{src/adm2/calltables.i
  &PARAM-TABLE-TYPE = "1"
  &PARAM-TABLE-NAME = "ttCallParam"}

/* Create the parameter values into the parameter value table */
CREATE ttCallParam.
ASSIGN
  ttCallParam.iParamNo    = 1
  ttCallParam.cDataType   = "CHARACTER"
  ttCallParam.cIOMode     = "INPUT"
  ttCallParam.cCharacter  = "aaa"
.
CREATE ttCallParam.
ASSIGN
  ttCallParam.iParamNo    = 2
  ttCallParam.cDataType   = "INTEGER"
  ttCallParam.cIOMode     = "INPUT-OUTPUT"
  ttCallParam.iInteger    = 12
.
CREATE ttCallParam.
ASSIGN
  ttCallParam.iParamNo    = 3
  ttCallParam.cDataType   = "HANDLE"
  ttCallParam.cIOMode     = "OUTPUT"
  ttCallParam.hHandle     = ?
.

RUN adm2/calltable.p
("emptyProcParam",      /* procedure to call */
 "calls.p",             /* containing procedure file */
 INPUT TEMP-TABLE ttCallParam:HANDLE, /* Handle of temp table */
 INPUT-OUTPUT TABLE-HANDLE htt). /* unknown - no table handle */
```

After the call completes, the return values are available in output parameter records. In [Example A-3](#), records 2 and 3 contain the output parameters from the call.

In addition, two extra records are appended to the table. The field `cParamName` on these fields is set to **callReturnValue** and **errReturnValue**. The **callReturnValue** field contains the value of the return value from the function or procedures and the **errReturnValue** field contains a value if an error condition occurs.

The code in [Example A-4](#) illustrates using `calltable.p` with additional requirements for mappings. The difference between [Example A-3](#) and [Example A-4](#) is that the API is now forced to instantiate `calls.p` and obtain the signature to `emptyProcParam` to complete the mapping of values from **ttCallParam** to the values in the procedure. This is more expensive in terms of performance, but does however provide some options for specifying parameters.

As in [Example A-3](#), the code in [Example A-4](#) appends two extra records to the table after execution of the call. The field `cParamName` on these fields is set to **callReturnValue** and **errReturnValue**. The **callReturnValue** field contains the value of the return value from the function or procedure, and the **errReturnValue** field contains a value if an error condition occurs.

Example A-4: Using `calltable.p` to invoke a call and to instantiate `calls.p`

```

DEFINE VARIABLE hTT AS HANDLE NO-UNDO.

/* Include the temp-table definition */
{src/adm2/calltables.i
 &PARAM-TABLE-TYPE = "4"
 &PARAM-TABLE-NAME = "ttCallParam"}

/* Create the parameter values into the parameter value table */
CREATE ttCallParam.
ASSIGN
    ttCallParam.cParamName = "pcChar"
    ttCallParam.cValue     = "aaa"
.
CREATE ttCallParam.
ASSIGN
    ttCallParam.cParamName = "piInt"
    ttCallParam.cValue     = "12"
.
CREATE ttCallParam.
ASSIGN
    ttCallParam.cParamName = "phHnd1"
    ttCallParam.cValue     = ?
.

RUN adm2/calltable.p
("emptyProcParam",      /* procedure to call */
 "calls.p",             /* containing procedure file */
 INPUT TEMP-TABLE ttCallParam:HANDLE, /* Handle of temp table */
 INPUT-OUTPUT TABLE-HANDLE hTT). /* unknown - no table handle */

```

calltablett.p

The `calltablett.p` single-point entry call provides the functionality of `calltable.p` along with the ability to pass temp-tables. This is similar to using [“callstringtt.p.”](#) As with `callstringtt.p`, you can use the `callttparam.i` include file to simplify passing temp-tables. The code in [Example A-5](#) illustrates the call done using the `callstringtt.p` call and `calltablett.p`.

Example A-5: Using `callstringtt.p` to make a call

```
DEFINE VARIABLE hTT AS HANDLE EXTENT 64 NO-UNDO.

/* Include the temp-table definition */
{src/adm2/calltables.i
 &PARAM-TABLE-TYPE = "4"
 &PARAM-TABLE-NAME = "ttCallParam"}

/* Create the parameter values into the parameter value table */
CREATE ttCallParam.
ASSIGN
    ttCallParam.cParamName = "phCustomerTempTable"
    ttCallParam.cValue      = "T:01"
.
CREATE ttCallParam.
ASSIGN
    ttCallParam.cParamName = "pcSessionID"
    ttCallParam.cValue      = cSessionID
.
CREATE ttCallParam.
ASSIGN
    ttCallParam.cParamName = "phOrderBuffer"
    ttCallParam.cValue      = "B:02"
.

hTT[01] = TEMP-TABLE ttCust:HANDLE.
hTT[02] = TEMP-TABLE ttOrder:HANDLE

RUN adm2/calltablett.p
(("obtainCustomerData", /* call to make */
 "customerbl.p",        /* manager to use */
 INPUT TEMP-TABLE ttCallParam:HANDLE, /* Handle of temp table */
 INPUT-OUTPUT TABLE-HANDLE hTT,    /* unknown - no table handle */
 "",                               /* Tables to skip */
 {src/adm2/callttparam.i
  &ARRAYFIELD = "hTT"}
).
```


Temp-table include files

The include files contain definitions for temp-tables that you can use to provide call parameters. The following sections provide information about these include files:

- [calltables.i](#)
- [callttparam.i](#)

calltables.i

The `calltables.i` include file located in the `src/adm2/` directory contains the definition of four temp-tables that provide call parameters to the **setupTTFrom** functions. The following list, presented in order of efficiency, shows how tables provide call parameters to the `setupTTFrom` functions:

1. Parameters stored by parameter position where the value is in the native data type.
2. Parameters stored by parameter position where the value is stored in a character field.
3. Parameters stored by parameter name where the value is stored in the native data type.
4. Parameters stored by parameter name where the value is stored in a character field.

When storing parameters by position, the parameters are added to the temp-table in the order in which they are specified. You must specify all parameters. This form of the temp-table requires that the Parameter Holder temp-table structure be built using `setupTTFromTable`.

When storing parameters by name, the parameter name specified must match the name of the parameter as it is retrieved from a GET-SIGNATURE call on the internal entry being invoked. This temp-table format requires that the Parameter Holder temp-table structure be built using `setupTTFromSig`.

This include file takes the following optional parameters:

PARAM-TABLE-TYPE

Indicates the type of table to be used. Defaults to 1.

PARAM-TABLE-NAME

A name to apply to the temp-table. Defaults to **ttCallParam**.

For more information about temp-table types, see the “[Temp-table types](#)” section.

callttparam.i

Although most AppServer calls require that no more than two or three temp-tables be passed between the AppServer and the client, some calls might require a larger number of temp-tables. For example, the Dynamics Repository Manger FetchObject call or a call to a SmartBusinessObject can require 20 or more temp-tables. In this instance, the `callttparam.i` include file provides an easy way to include the temp-table handles required by `callstringtt.p`. For more information, see the “[callstringtt.p](#)” and “[calltables.i](#)” sections.

Before using the `callttparam.i` file, you must define an array of handles. The following describes the parameters supplied in the `callttparam.i`:

ARRAYFIELD

Required parameter that names an extent 64 variable of data type handle.

MODE

Optional parameter. One of INPUT, OUTPUT, or INPUT-OUTPUT. The default is INPUT-OUTPUT. The MODE applies to all 64 parameters. You cannot override the mode for individual parameters.

T01 to T64

Optional parameter. An include file parameter that evaluates to a valid 4GL expression that follows the MODE. If not specified, the default is

“TABLE-HANDLE {&ARRAYFIELD}[subscript]”. This form lets you specify each table differently. For example, **&T01 = “TABLEttCustomer”, &T02 - “TABLE-HANDLEhTOrder”**.

In [Example A–6](#), the temp-tables ttCust, ttOrder, and a dynamic temp-table with the handle httTest are being passed and the remaining 61 are empty. The three temp-tables being passed are passed differently. Note the following in the example code:

- An array (hTT) of 64 handles is defined and used with the include file. The array is reset so that all elements are the unknown value. This is done so that when no temp-table is provided, nothing is sent.
- In the lines of code after the array is cleared, table 2 and table 3 are set to the value of the handles for the ttOrder and httTest temp-tables.

- In the call that uses the include file, the array variable is specified as the variable for the &ARRAYFIELD include parameter.
- The code automatically passes the array field with the other two temp-tables. The &T01 parameter passes the ttCust temp-table using a TABLE rather than a TABLE-HANDLE parameter. Therefore, the only include parameter is &T01.

Example A-6: Using the callttparam.i include file

```

DEFINE VARIABLE i AS INTEGER NO-UNDO.
DEFINE VARIABLE hTT AS HANDLE EXTENT 64 NO-UNDO.

/* some code */

/* make sure array has unknowns for all handles. */
DO i = 1 TO 64:
    hTT[i] = ?.
END.

/* Table 2 must be the Order temp-table and table 3 is the test table */
hTT[02] = TEMP-TABLE ttOrder:HANDLE. /* Handle Order temp-table */
hTT[03] = httTest. /* Handle to temp-table test */

RUN adm2/callstringtt.p
    ("obtainCustomerData", /* call to make */
     "customerbl.p", /* manager to use */

/* Now map the parameters to the table handles */
    "INPUT HANDLE 'T:01', INPUT CHARACTER cSessionID, INPUT HANDLE 'B:02', INPUT
    HANDLE 'T:03'", /* parameter list */

    "", /* Tables to skip */

/* Pass in all 64 table handles. 1 is the table structure. */
    {src/adm2/callttparam.i
     &ARRAYFIELD = "hTT"
     &T01 = "TABLE ttCust" }
    ).

```

Temp-table types

This section describes the different temp-tables types that you can use to provide call parameters for the `setupTTFrom` functions. The different temp-table types are:

- [Position native data type table](#)
- [Position character table](#)
- [Name native data type table](#)
- [Name character table](#)

Position native data type table

The Position Native data type table (table type 1), contains a record for each parameter with a sequence number that matches the ordinal position of the parameter in the call. The value of the parameter is stored in one of the value fields in the native data type of the parameter.

[Figure A–1](#) shows the definition of the native position data type table.

```
DEFINE TEMP-TABLE ttCallParam NO-UNDO
  FIELD iParamNo      AS INTEGER
  FIELD cParamName    AS CHARACTER
  FIELD cDataType     AS CHARACTER
  FIELD cIOMode       AS CHARACTER
  FIELD cCharacter     AS CHARACTER
  FIELD tDate         AS DATE
  FIELD lLogical      AS LOGICAL
  FIELD iInteger       AS INTEGER
  FIELD dDecimal      AS DECIMAL
  FIELD aRaw          AS RAW
  FIELD hHandle       AS HANDLE
  FIELD rRowid        AS ROWID
  INDEX pudx IS UNIQUE PRIMARY
    iParamNo.
```

Figure A–1: Position native data type table definition

In [Example A-7](#), the following parameters are added to the ttCallParam temp-table:

- A character input parameter with a value of aaa.
- An integer input-output parameter with a value of 12.
- A handle output parameter.

Example A-7: Parameters stored in a position native data type table

```
/* Include the temp-table definition */
{src/adm2/calltables.i
 &PARAM-TABLE-TYPE = "1"}

/* Create the parameter values into the parameter value table */
CREATE ttCallParam.
ASSIGN
    ttCallParam.iParamNo    = 1
    ttCallParam.cDataType   = "CHARACTER"
    ttCallParam.cIOMode     = "INPUT"
    ttCallParam.cCharacter  = "aaa"
.
CREATE ttCallParam.
ASSIGN
    ttCallParam.iParamNo    = 2
    ttCallParam.cDataType   = "INTEGER"
    ttCallParam.cIOMode     = "INPUT-OUTPUT"
    ttCallParam.iInteger    = 12
.
CREATE ttCallParam.
ASSIGN
    ttCallParam.iParamNo    = 3
    ttCallParam.cDataType   = "HANDLE"
    ttCallParam.cIOMode     = "OUTPUT"
    ttCallParam.hHandle     = ?
.
```

The parameter number specified in **iParamNo** indicates the order in which the parameters are specified. As a result, a RUN statement using these parameters would look something like the code in [Example A-8](#). Note the order in which the parameters appear.

Example A-8: Parameters specified in iParamNo

```
DEFINE VARIABLE iVal    AS INTEGER NO-UNDO.
DEFINE VARIABLE hHandle AS HANDLE NO-UNDO.
iVal = 12.
hHandle = ?.

RUN x.p (INPUT "aaa", INPUT-OUTPUT iVal, OUTPUT hHandle).
```

Position character table

The Position Character table (table type 2), provides the ordinal position of the parameter and stores the parameter value in a character field. As with the position native data type table, the ordinal position of the parameter is important.

Figure A–2 shows the definition of the position character type temp-table.

```
DEFINE TEMP-TABLE ttCallParam NO-UNDO
  FIELD iParamNo      AS INTEGER
  FIELD cParamName    AS CHARACTER
  FIELD cDataType     AS CHARACTER
  FIELD cIOMode       AS CHARACTER
  FIELD cValue        AS CHARACTER
  INDEX pudx IS UNIQUE PRIMARY
    iParamNo.
```

Figure A–2: Position character table type definition

The code in [Example A–9](#) illustrates how to set up the same set of parameters as those in [Example A–7](#) for the position native data type table.

Example A–9: Parameters stored in a position character table type

```
/* Include the temp-table definition */
{src/adm2/calltables.i
  &PARAM-TABLE-TYPE = "2"}

/* Create the parameter values into the parameter value table */
CREATE ttCallParam.
ASSIGN
  ttCallParam.iParamNo    = 1
  ttCallParam.cDataType   = "CHARACTER"
  ttCallParam.cIOMode     = "INPUT"
  ttCallParam.cValue      = "aaa"
.
CREATE ttCallParam.
ASSIGN
  ttCallParam.iParamNo    = 2
  ttCallParam.cDataType   = "INTEGER"
  ttCallParam.cIOMode     = "INPUT-OUTPUT"
  ttCallParam.cValue      = "12"
.
CREATE ttCallParam.
ASSIGN
  ttCallParam.iParamNo    = 3
  ttCallParam.cDataType   = "HANDLE"
  ttCallParam.cIOMode     = "OUTPUT"
  ttCallParam.cValue      = ?
.
```

The conversion to the appropriate data type is done when the dynamic temp-table structure is built from this table.

Name native data type table

With a Name Native data type table (table type 3), the caller needs to know what the name of the parameter is that is being set. This temp-table type requires the use of the `setupTTFromSig` call to build the temp-table required to make the call.

[Figure A-3](#) shows the definition of the Name native data type temp-table.

```
DEFINE TEMP-TABLE ttCallParam NO-UNDO
  FIELD cParamName      AS CHARACTER
  FIELD cCharacter      AS CHARACTER
  FIELD tDate           AS DATE
  FIELD lLogical        AS LOGICAL
  FIELD iInteger        AS INTEGER
  FIELD dDecimal        AS DECIMAL
  FIELD aRaw            AS RAW
  FIELD hHandle         AS HANDLE
  FIELD rRowid          AS ROWID
  INDEX pudx IS UNIQUE PRIMARY
    cParamName.
```

Figure A-3: Name native data type table definition

The code in [Example A–10](#) illustrates how to set up a Name native data type table (table type 3).

Example A–10: Name native data type table

```
/* Include the temp-table definition */
{src/adm2/calltables.i
  &PARAM-TABLE-TYPE = "3"}

/* Create the parameter values into the parameter value table */
CREATE ttCallParam.
ASSIGN
  ttCallParam.cParamName = "pcChar"
  ttCallParam.cCharacter = "aaa"
.
CREATE ttCallParam.
ASSIGN
  ttCallParam.cParamName = "piInt"
  ttCallParam.iInteger   = 12
.
CREATE ttCallParam.
ASSIGN
  ttCallParam.cParamName = "phHndl"
  ttCallParam.hHandle    = ?
.
```

Notes: There is no need to specify the mode or data type of the parameter as these are derived from the signature of the call and placed in the appropriate field. However, when using this table type, the value of the parameter being set must be set in the correct data type field, otherwise the value is never retrieved.

Name character table

The Name Character table (table type 4) is similar to the Name native data type table. However, in the Name character table, the value of the parameter is provided in character format rather than in the native data type. [Example A-11](#) illustrates the definition of the Name Character table.

Example A-11: Name character table definition

```
DEFINE TEMP-TABLE ttCallParam NO-UNDO
  FIELD cParamName      AS CHARACTER
  FIELD cValue          AS CHARACTER
  INDEX pudx IS UNIQUE PRIMARY
    cParamName.
```

The following example illustrates setting up data in the name character table type:

```
/* Include the temp-table definition */
{src/adm2/calltables.i
  &PARAM-TABLE-TYPE = "4"}

/* Create the parameter values into the parameter value table */
CREATE ttCallParam.
ASSIGN
  ttCallParam.cParamName = "pcChar"
  ttCallParam.cValue     = "aaa"
.
CREATE ttCallParam.
ASSIGN
  ttCallParam.cParamName = "piInt"
  ttCallParam.cValue     = "12"
.
CREATE ttCallParam.
ASSIGN
  ttCallParam.cParamName = "phHnd1"
  ttCallParam.cValue     = ?
.
```

Invoking the dynamics call wrapper at the API level

The call wrapper consists of several procedure files with most of the code contained in `caller.p`. The `caller.p` file is derived from `src/adm2/smart.p`, runs as a persistent procedure and is called as a library. A template procedure is provided as a starting point.

A call is broken down into the following phases:

- 1 ♦ **Setup** — consists of creating the Parameter Holder temp-table and populating it with new data. The Parameter Holder temp-table is created by a call to one of these functions:
 - `setupTTFromSig`
 - `setupTTFromString`
 - `setupTTFromTable`
- 2 ♦ **Invocation** — Before a call can be invoked, a DYNAMIC CALL object is created and properties of the call object are set. The values for these properties are derived from the `ttCall` temp-table that is defined in the definitions section of `caller.p`. `ttCall` is local to `caller.p` and is not exposed in any form to external procedures. When a call is added to `ttCall`, it is allocated a call number and this is passed as a parameter to the `invoke` procedure. The values in the `ttCall` temp-table record are set by a call during the `invokeCall` procedure from the input parameters to the procedure. As a result, all the attributes of the DYNAMIC CALL handle can be derived from the `ttCall` temp-table.
- 3 ♦ **Cleanup** — Deletes the `ttCall` record and the following handles:
 - The handle to the call object.
 - The persistent procedure handle if the procedure was instantiated by the dynamics call.
 - The handle to the asynchronous call object.
 - The handle to the Parameter Holder temp-table.

Figure A-4 shows the steps you must make to invoke a call at the API level.

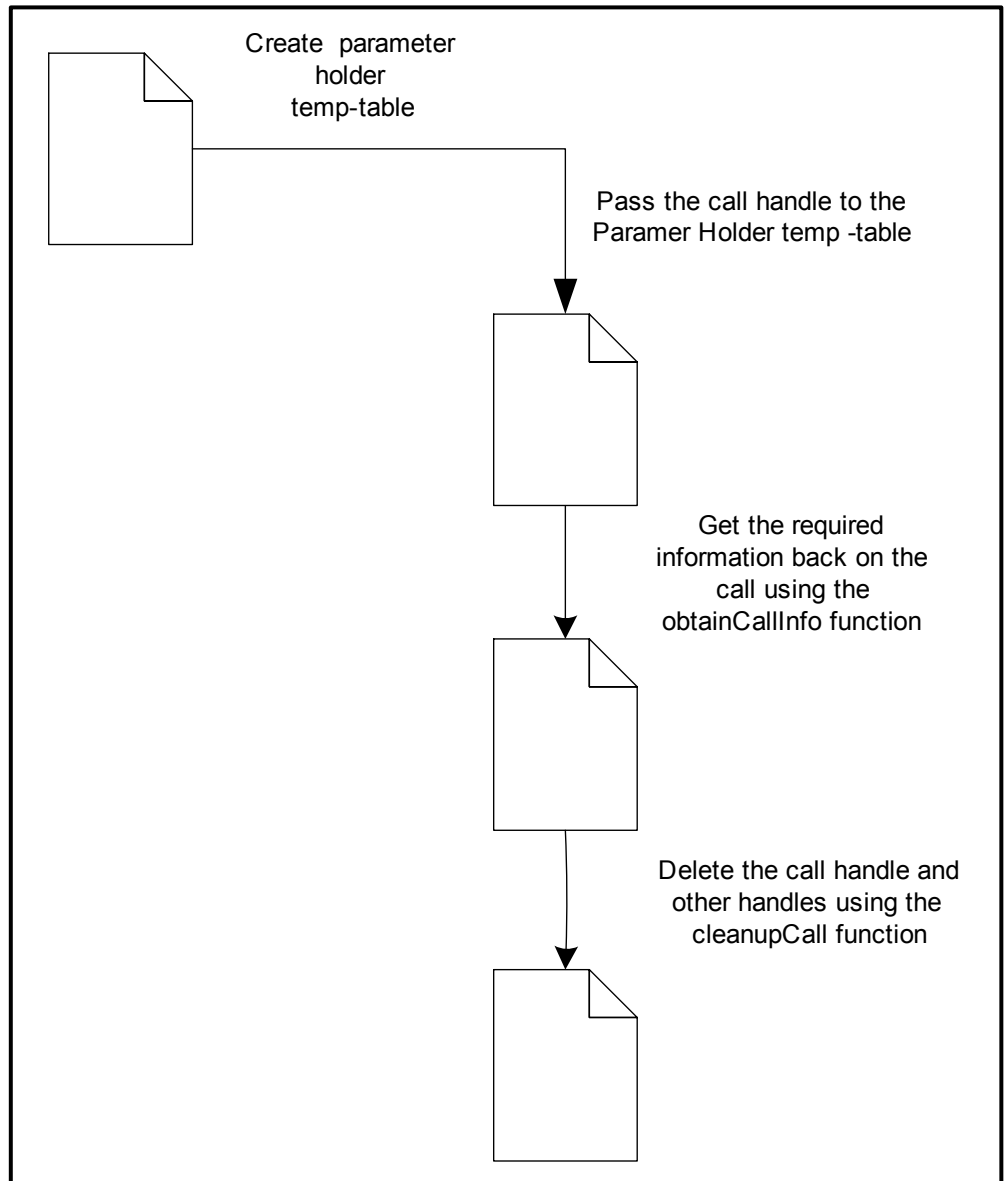


Figure A-4: API calls required to invoke a dynamics call

The calls shown in [Figure A–4](#) illustrate the API calls you need to make to invoke a call:

- 1 ♦ Create the Parameter Holder temp-table from the parameters using one of the following: `setupTTFromSig`, `setupTTFromString`, or `setupTTFromTable`.
- 2 ♦ Pass the handle to the Parameter Holder temp-table created in the [Step 1](#), along with all the other parameters to `invokeCall`.
- 3 ♦ Use the `obtainCallInfo` function call to get the required information back on the call. The Parameter Holder temp-table created in [Step 1](#) also contains output parameters and return values.
- 4 ♦ Use `cleanupCall` to delete the call handle and any other handles that were created during the `invokeCall`. Note that it is possible to only delete certain of the handles.

These steps are illustrated in code [Example A–12](#) through [Example A–15](#). Each of the examples makes a call to the internal procedure `PROCEDURE emptyProcParam`.

Invoking the dynamics call wrapper at an API level requires that you write more code and is more complicated to implement. However, this method of invoking the dynamic call wrapper provides enhanced performance and more control of error handling and the call handle.

For more information about the APIs used to invoke the dynamics call wrapper, see the “[API Reference](#)” section.

The code in [Example A–12](#) illustrates the steps necessary to make dynamics calls.

Example A–12: Steps for making dynamics calls

```
PROCEDURE emptyProcParam:
  DEFINE INPUT      PARAMETER pcChar  AS CHARACTER  NO-UNDO.
  DEFINE INPUT-OUTPUT PARAMETER piInt  AS INTEGER    NO-UNDO.
  DEFINE OUTPUT     PARAMETER phHndl  AS HANDLE     NO-UNDO.
  piInt = 35.
  phHndl = THIS-PROCEDURE.
  RETURN "Hello":U.
END.
```

The Parameter Holder temp-table contains a field for each parameter you want to pass. The Parameter Holder temp-table structure contains a field for each parameter and two extra fields: **callReturnValue** that contains the return value from the function or procedure and **errReturnValue** that contains a value only when an error occurs. The data types for the fields are the same as the data type of the parameter being referenced. The COLUMN-LABEL for the field contains the data type to be used for the call and the mode of the parameter. For example, INPUT, INPUT-OUTPUT, OUTPUT, or OUTPUT-APPEND.

If source parameters are provided to the setupTTFrom call, the initial values of the temp-table fields are set to the values of the source parameters so that it is not necessary to create a temp-table record in the Parameter Holder temp-table until the call is made. When the record is created, the initial value is derived from whatever was set.

The code in [Example A-13](#) builds the parameters into a temp-table that is then used to create a parameter temp-table.

Example A-13: Creating the parameter holder temp-table

(1 of 2)

```
/* example 1 */

/* Include the temp-table for the call parameters */
{src/adm2/calltables.i}

/* Run the target procedure persistently */
RUN calls.p PERSISTENT SET hProc.

/* Create a record for each parameter */
DO TRANSACTION:
  CREATE ttCallParam.
  ASSIGN
    ttCallParam.iParamNo = 1
    ttCallParam.cDataType = "CHARACTER"
    ttCallParam.cIOMode = "INPUT"
    ttCallParam.cCharacter = "aaa"
  .
  CREATE ttCallParam.
  ASSIGN
    ttCallParam.iParamNo = 2
    ttCallParam.cDataType = "INTEGER"
    ttCallParam.cIOMode = "INPUT-OUTPUT"
    ttCallParam.iInteger = 12
  .
  CREATE ttCallParam.
  ASSIGN
    ttCallParam.iParamNo = 3
    ttCallParam.cDataType = "HANDLE"
    ttCallParam.cIOMode = "OUTPUT"
    ttCallParam.hHandle = ?
  .
END.
```

Example A-13: Creating the parameter holder temp-table

(2 of 2)

```

/* Step 1. Create a Parameter Holder temp-table for the call */
hTable = DYNAMIC-FUNCTION("setupTTFromTable" IN THIS-PROCEDURE,
    "ttParameters", /* Temp-table name */
    "CHARACTER", /* Return value data type */
    TEMP-TABLE ttCallParam:HANDLE).

/* Step 2. Invoke the call */
RUN invokeCall IN THIS-PROCEDURE
    ("emptyProcParam", /* Name of internal procedure */
    hProc, /* Handle of the IN procedure */
    PROCEDURE-CALL-TYPE, /* We're calling a procedure */
    hTable, /* Handle from setupTTFromTable */
    NO, /* Persistent */
    ?, /* Server */
    NO, /* Asynch */
    "", /* Event procedure for Asynch */
    ?, /* Event procedure context */
    OUTPUT iCall). /* Call number */

/* Step 3. Obtain information about the call */
hCall = obtainCallInfo(iCall, OUTPUT cReturnValue, OUTPUT hTable).

/* process the call information */
/* Step 4. Cleanup.
   Delete all the call information except the Parameter Holder
   temp-table */
RUN cleanupCall IN THIS-PROCEDURE (iCall, "!T,*").

DELETE OBJECT hTable.

```

The code in [Example A-14](#) illustrates how parameters are derived from the signature and then set.

Example A-14: Deriving parameters from the signature

```

/* Run the target procedure persistently */
RUN calls.p PERSISTENT SET hProc.

/* Step 1. Create a Parameter Holder temp-table for the call
   from the signature of the call that we want to make */
hTable = DYNAMIC-FUNCTION("setupTTFromSig" IN THIS-PROCEDURE,
    "ttParameters", /* Temp-table name */
    hProc,          /* Persistent procedure */
    "emptyProcParam", /* Internal procedure */
    "",             /* Signature */
    ? ).            /* Handle to TT with initial values */

/* Now set the parameter values. */
hTable:DEFAULT-BUFFER:BUFFER-CREATE().
hTable:DEFAULT-BUFFER:BUFFER-FIELD("pcChar"):BUFFER-VALUE = "aaa".
hTable:DEFAULT-BUFFER:BUFFER-FIELD("pcInt"):BUFFER-VALUE = 12.
hTable:DEFAULT-BUFFER:BUFFER-FIELD("phHnd1"):BUFFER-VALUE = ?.
hTable:DEFAULT-BUFFER:BUFFER-RELEASE().

/* Step 2. Invoke the call */
RUN invokeCall IN THIS-PROCEDURE
    ("emptyProcParam", /* Name of internal procedure */
    hProc,             /* Handle of the IN procedure */
    PROCEDURE-CALL-TYPE, /* We're calling a procedure */
    hTable,            /* Handle from setupTTFromTable */
    NO,                /* Persistent */
    ?,                 /* Server */
    NO,                /* Asynch */
    "",               /* Event procedure for Asynch */
    ?,                /* Event procedure context */
    OUTPUT iCall).     /* Call number */

/* Step 3. Obtain information about the call */
hCall = obtainCallInfo(iCall, OUTPUT cReturnValue, OUTPUT hTable).

/* process the call information */

/* Step 4. Cleanup */
RUN cleanupCall IN THIS-PROCEDURE (iCall, "").

```

In [Example A–15](#), the parameters are derived from a string that is passed in to create the dynamics temp-table.

Example A–15: Deriving parameters from a string to create a dynamics temp-table

```
/* Run the target procedure persistently */
RUN calls.p PERSISTENT SET hProc.

/* Step 1. Create a Parameter Holder temp-table for the call
   from the string that we pass into the call */
hTable = DYNAMIC-FUNCTION("setupTTFromString" IN THIS-PROCEDURE,
    "ttParameters", /* Temp-table name */
    "CHARACTER", /* RETURN-VALUE data type */
    "INPUT CHARACTER 'aaa', INPUT INTEGER '12', INPUT HANDLE '?'").
/* Parameter string */

/* Step 2. Invoke the call */
RUN invokeCall IN THIS-PROCEDURE
("emptyProcParam", /* Name of internal procedure */
 hProc, /* Handle of the IN procedure */
 PROCEDURE-CALL-TYPE, /* We're calling a procedure */
 hTable, /* Handle from setupTTFromTable */
 NO, /* Persistent */
 ?, /* Server */
 NO, /* Asynch */
 "", /* Event procedure for Asynch */
 ?, /* Event procedure context */
 OUTPUT iCall). /* Call number */

/* Step 3. Obtain information about the call */
hCall = obtainCallInfo(iCall, OUTPUT cReturnValue, OUTPUT hTable).
/* process the call information */

/* Step 4. Cleanup */
RUN cleanupCall IN THIS-PROCEDURE (iCall, "").
```

API Reference

This section provides details about the APIs for the Dynamics Call Wrapper and explains how each API procedure works.

callstring.p Procedure

External procedure that allows for a single RUN statement to invoke a call using a string that defines the parameters. The procedure is useful for minimizing the number of requests it takes to invoke a call on the AppServer.

All outputs and return values from the invoked procedure are available through the properties or parameters supplied as parameters.

Parameters:

`pcCallName` INPUT CHARACTER

Name of an external or internal procedure or function to be invoked.

`pcTarget` INPUT CHARACTER

The name of a manager procedure, the filename of a relatively or absolutely pathed procedure, or an integer value that evaluates to a procedure handle. If the value of this parameter is "" or ?, by default, `pcCallName` contains the name of a procedure that is to be run nonpersistently.

The parameter is optional. If nothing is specified, then "" or ? is passed.

`pcTargetFlags` INPUT CHARACTER

This parameter can contain modifiers that are used to invoke the persistent procedure. A modifier can be a valid combination of the following:

- **P(ersistent)** — Indicates that a new instance of the procedure should be instantiated persistently and left running.
- **A(DM2)** — Indicates that a new instance of an ADM2 procedure should be invoked persistently and the `initializeObject` internal procedure called to initialize it. The ADM2 procedure is left running after the call is complete.
- **S(ingle)** — Indicates that a new instance of the procedure should be instantiated if a running version is not found and left running.
- **K(ill)** — Indicates that if a procedure was instantiated during the call, it should be deleted before control is returned.

The default is to apply the behavior of the **S** parameter. That is, a persistent procedure is started if it is not found by walking the procedure stack and left running after the call is complete.

You can specify any of the **P**, **A**, or **S** modifiers in combination with **K** to shut down the procedure. For example, **PK** instructs the caller to instantiate a new instance of the procedure persistently and delete it when it is complete.

The parameter is optional. If nothing is specified, then "" is passed.

pcCallParmString INPUT CHARACTER

A string that contains the parameters to pass to the procedure or function that is being invoked. The string is a comma-separated list of parameters. Each parameter is a string consisting of space-delimited values in the form "*mode data type parameter*" where:

- *mode* is one of INPUT, OUTPUT, INPUT-OUTPUT, or OUTPUT-APPEND.
- *data type* is one of CHARACTER, DATE, LOGICAL, INTEGER, DECIMAL, RAW, HANDLE, or ROWID.
- *parameter* is the name of either a property that was previously set using setPropertyList or setSessionParam, or a single quoted constant. If a property is specified, the procedure attempts to evaluate the property value by calling getPropertyList. If no property is available, or Dynamics is not running, a call is made to getSessionParam for the property value. If neither call succeeds, the unknown value is passed.

If the mode of the parameter is either OUTPUT or INPUT-OUTPUT, the property that is specified contains the output value from the call after the call is complete.

The parameter is optional. If nothing is specified, then "" or ? is passed.

Notes:

- Once the procedure finishes executing, all temporary handles are deleted and the ttCall record no longer exists for the call. As a result, the Dynamics Call Wrapper has no information about this call so that there are no memory leaks created by calling callstring.p.
- If the mode of the parameter is OUTPUT or INPUT-OUTPUT and a property that was set using setPropertyList is specified, the value of the property is set on the server side. It is not synchronized with the client unless the entire call is executed on the client side.

callstringtt.p procedure

External procedure that allows a single RUN statement to invoke a call using a string that defines the parameters and pass up to 64 temp-tables. This procedure is useful for minimizing the number of request it takes to invoke a call on the AppServer.

All outputs and return values from the invoked procedure are available through the properties or parameters supplied as parameters.

Parameters:

pcCallName INPUT CHARACTER

Name of an external or internal procedure or function to be invoked.

pcTargetObjec INPUT CHARACTER

Name of a manager procedure, the filename of a relatively or absolutely pathed procedure, or an integer value that evaluates to a procedure handle. If the value of this parameter is "" or ?, by default, pcCallName contains the name of a procedure that is to be run nonpersistently.

The parameter is optional. If nothing is specified, then "" or ? is passed.

pcTargetFlags INPUT CHARACTER

This parameter can contain modifiers that are used to invoke the persistent procedure. A modifier can be a valid combination of the following:

- **P(ersistent)** — Indicates that a new instance of the procedure should be instantiated persistently and left running.
- **A(DM2)** — Indicates that a new instance of an ADM2 procedure should be invoked persistently and the initializeObject internal procedure called to initialize it. The ADM2 procedure is left running after the call is complete.
- **S(ingle)** — Indicates that a new instance of the procedure should be instantiated if a running version is not found and left running.
- **K(ill)** — Indicates that if a procedure was instantiated during the call, it should be deleted before control is returned. The default is to apply the behavior of the **S** parameter. As a result, a persistent procedure is started if it is not found by walking the procedure stack and is left running after the call is complete.

You can specify any of the **P**, **A**, or **S** modifiers in combination with **K** to shut down the procedure. For example, **PK** instructs the caller to instantiate a new instance of the procedure persistently and delete it when it is complete.

The parameter is optional. If nothing is specified, then "" is passed

pcCallParmString INPUT CHARACTER

A string containing the parameters to pass to the procedure or function that is being invoked. The string is a comma-separated list of parameters. Each parameter is a string consisting of space-delimited values in the form "*mode data type parameter*" where:

- *mode* is one of INPUT, OUTPUT, INPUT-OUTPUT, or OUTPUT-APPEND.
- *data type* is one of CHARACTER, DATE, LOGICAL, INTEGER, DECIMAL, RAW, HANDLE, TABLE-HANDLE, or ROWID.
- *parameter* is the name of either a property that was previously set using `setPropertyList` or `setSessionParam`, or a single quoted constant that does not contain spaces or commas. If a property is specified, the procedure attempts to evaluate the property value by calling `getPropertyList`. If no property is available, or Dynamics is not running, a call is made to `getSessionParam` for the property value. If neither call succeeds, the unknown value is passed.

If the mode of the parameter is either OUTPUT or INPUT-OUTPUT, the property that is specified contains the output value of the call after the call is complete. If the *data type* is either HANDLE or TABLE-HANDLE, the *parameter* can also be an integer value between 1 and 32 that maps to one of the tables passed in with the call.

The parameter is optional. If nothing is specified, then "" or ? is passed.

pcHandlesToSkip INPUT CHARACTER

By default, the call wrapper does a DELETE OBJECT on tables that are listed in `phCallTableHandle01` through 64 before returning. This avoids memory leaks caused by the duplication of temp-tables. In some cases, this behavior might be undesirable, such as when the table being returned is a dynamics temp-table that should be retained in the cache on the server.

To address this issues, this parameter allows a comma- separated list of numbers between 1 and 64 corresponding to the handles below. If a number is found in the list, the corresponding handle is not deleted in the procedure prior to control returning to the caller.

NOTE: Using an * in a CAN-DO list indicates that none of the handles should be deleted.

The parameter is optional. If nothing is specified, then "" is passed.

phCallTableHandle01 to phCallTableHandle64 INPUT-OUTPUT TABLE-HANDLE

A table-handle that needs to be passed into the call. The parameter is optional. If nothing is specified, then ? is passed.

Notes:

- Once the procedure finishes executing, all temporary handles are deleted and the ttCall record no longer exists for the call. As a result, the Dynamic Call Wrapper has no information about this call so that there are no memory leaks created by calling callstring.p.
- The caller is responsible for cleaning up phCallParmTT and phCallParmTH and the phCallTempTable or phCallTableHandle objects.
- When execution passes to callstringtt.p, the procedure creates an array of 64 handles and stores the values of the handles passed into the array so that temp-table copying is done only when required to invoke the call.
- If the mode of the parameter is OUTPUT or INPUT-OUTPUT and a property that was set using setPropertyList is specified, the value of the property is set on the server side. It is not synchronized with the client unless the entire call is executed on the client side.

calltable.p procedure

External procedure that allows for a single RUN statement to invoke a call using a temp-table that defines the parameter values. The temp-table can be in any of the formats supported by the src/adm2/calltables.i include file. If a temp-table of type 1 or 2 is used, the parameters to the called object are not type checked. If a temp-table of type 3 or 4 is used, the call must be to a function or an internal procedure in a persistent procedure, and the signature is checked against the internal entries before the call is made. All outputs and return values from the invoked procedure are available through the appropriate rows on the returned temp-table.

Parameters:

pcCallName INPUT CHARACTER

The name of an external or internal procedure or function to invoke.

pcTarget INPUT CHARACTER

Name of a manager procedure, the filename of a relatively or absolutely pathed procedure, or an integer value that evaluates to a procedure handle. If the value of this parameter is "" or ?, by default, pcCallName contains the name of a procedure that is to run nonpersistently.

The parameter is optional. If nothing is specified, then "" or ? is passed.

pcTargetFlags INPUT CHARACTER

This parameter can contain modifiers that are used to invoke the persistent procedure. A modifier can be a valid combination of the following:

- **P(ersistent)** — Indicates that a new instance of the procedure should be instantiated persistently and left running.
- **A(DM2)** — Indicates that a new instance of an ADM2 procedure should be invoked persistently and the initializeObject internal procedure called to initialize it. The ADM2 procedure is left running after the call is complete.
- **S(ingle)** — Indicates that a new instance of the procedure should be instantiated if a running version is not found and left running.
- **K(ill)** — Indicates that if a procedure was instantiated during the call, it should be deleted before control is returned.

The default is to apply the behavior of the **S** parameter. That is, a persistent procedure is started if it is not found by walking the procedure stack and it is left running after the call is complete.

You can specify any of the **P**, **A**, or **S** modifiers in combination with **K** to shut down the procedure. For example, **PK** instructs the caller to instantiate a new instance of the procedure persistently and delete it when it is complete.

This parameter is optional. If nothing is specified, then "" is passed.

phCallParmTT INPUT HANDLE

Handle to a temp-table that could be either of the temp-tables defined in `adm2/calltables.i` or a temp-table that was previously created by a call to one of the `setupTTFrom` functions. The output values from the call are written back into this table after the call has been invoked. If this parameter is not specified and there are parameters to the procedure or function, the parameters have to be specified in the `phCallParmTH` parameter.

This parameter is optional. If nothing is specified, then ? is passed.

phCallParmTH INPUT-OUTPUT TABLE-HANDLE

Similar to `phCallParmTT` except that a TABLE-HANDLE is passed to support AppServer calls. The table is passed as an INPUT-OUTPUT parameter so that the return values can be returned to the client.

This parameter is optional. If nothing is specified, then ? is passed.

Notes:

- Once the procedure finishes executing, all temporary handles are deleted and the ttCall record no longer exists for the call. As a result, the Dynamic Call Wrapper has no information about this call so that there are no memory leaks created by calling callstring.p.
- The caller is responsible for cleaning up phCallParmTT and phCallParmTH.
- If phCallParmTT is specified, it is used to determine the parameters to the call. Otherwise phCallParmTH is used. If neither are specified the call is attempted with no parameters.

calltablett.p procedure

External procedure that allows a single RUN statement to invoke a call using a temp-table that defines the parameter values. The temp-table can be in any of the formats supported by the src/adm2/calltables.i include file. If a temp-table of type 1 or 2 is used, the parameters to the called object are not type checked. If a temp-table of type 3 or 4 is used, the call has to be to a function or an internal procedure in a persistent procedure, and the signature is checked against the internal entries before the call is made. This variation of adm2/calltable.p supports the passing in of up to 64 temp-tables. For additional information, see the [“callstringtt.p”](#), [“calltable.p”](#), and [“calltables.i”](#) sections.

All outputs and return values from the invoked procedure are available through the rows in the returned temp-table.

Parameters:

pcCallNam INPUT CHARACTER

Name of an external or internal procedure or function to be invoked.

pcTarget INPUT CHARACTER

Name of a manager procedure, the filename of a relatively or absolutely pathed procedure, or an integer value that evaluates to a procedure handle. If the value of this parameter is "" or ?, by default, pcCallName contains the name of a procedure that is to be run nonpersistently.

This parameter is optional. If nothing is specified, then "" or ? is passed.

pcTargetFlags INPUT CHARACTER

This parameter can contain modifiers that are used to invoke the persistent procedure. A modifier can be a valid combination of the following:

- **P(ersistent)** — Indicates that a new instance of the procedure should be instantiated persistently and left running.
- **A(DM2)** — Indicates that a new instance of an ADM2 procedure should be invoked persistently and the initializeObject internal procedure called to initialize it. The ADM2 procedure is left running after the call is complete.
- **S(ingle)** — Indicates that a new instance of the procedure should be instantiated if a running version is not found and left running.
- **K(ill)** — Indicates that if a procedure was instantiated during the call, it should be deleted before control is returned.

The default is to apply the behavior of the **S** parameter. That is, a persistent procedure is started if it is not found by walking the procedure stack and it is left running after the call is complete.

You can specify any of the **P**, **A**, or **S** modifiers in combination with **K** to shut down the procedure. For example, **PK** instructs the caller to instantiate a new instance of the procedure persistently and delete it when it is complete.

This parameter is optional. If nothing is specified, then "" is passed.

pcHandlesToSkip INPUT CHARACTER

By default the call wrapper runs DELETE OBJECT on tables that are listed in phCallTableHandle01 through 64 before returning. This avoids memory leaks caused by the duplication of temp-tables. In some cases this behavior might be undesirable, such as when the table being returned is a dynamics temp-table that should be retained in the cache on the server.

To address this issue, this parameter allows a comma- separated list of numbers between 1 and 64 corresponding to the handles below. If a number is found in the list, the corresponding handle is not deleted in the procedure prior to returning control to the caller.

NOTE: Using an * in a CAN-DO list indicates that none of the handles should be deleted.

This parameter is optional. If nothing is specified, then "" is passed.

phCallParmTT INPUT HANDLE

Handle to a temp-table that could be either of the temp-tables defined in `adm2/calltables.i` or a temp-table that was previously created by a call to one of the `setupTTFrom` functions. The output values from the call are written back into this table after the call has been invoked. If this parameter is not specified and there are parameters to the procedure or function, the parameters have to be specified in the `phCallParmTH` parameter.

This parameter is optional. If nothing is specified, the `?` is passed.

phCallParmTH INPUT-OUTPUT TABLE-HANDLE

Similar to `phCallParmTT` except that a `TABLE-HANDLE` is passed to support `AppServer` calls. The table is passed as an `INPUT-OUTPUT` parameter so that the return values can be returned to the client.

This parameter is optional. If nothing is specified, then `?` is passed.

phCallTableHandle01 to phCallTableHandle64 INPUT-OUTPUTTABLE-HANDLE

Table-handle that needs to be passed into the call.

This parameter is optional. If nothing is specified, then `?` is passed.

Notes:

- Once the procedure has finished executing, all temporary handles are deleted and the `ttCall` record no longer exists for the call. As a result, the Dynamics Call Wrapper has no information about this call so that there are no memory leaks created by calling `callstring.p`.
- The caller is responsible for cleaning up `phCallParmTT`, `phCallParmTH`, and the `phCallTempTable` or `phCallTableHandle` objects.
- When the execution passes to `callstringtt.p`, the procedure creates an array of 64 handles and stores the values of the handles passed into the array. As a result, a temp-table is copied only when required to invoke the call.
- If `phCallParmTT` is specified, it is used to determine the parameters to the call. Otherwise `phCallParmTH` is used. If neither are specified, the call is attempted with no parameters.

cleanupCall procedure

Procedure used to delete any objects associated with a call that might be left inside the Dynamics Call Wrapper after the call finishes executing.

Parameters:

piCall INPUT INTEGER

A call number returned from a previous invokeCall request. This call number identifies the call you want cleaned up.

pcHandles INPUT CHARACTER

A comma-separated CAN-DO list used to indicate which handles to clean up. You can specify:

- **A** — Asynchronous request handle.
- **T** — Dynamics Temp-Table associated with call.
- **H** — Call Handle.

This parameter is optional. If nothing is specified, then "" is passed.

Notes:

- By default, all three handles are deleted when cleanupCall is invoked. To have all handles except the Call Handle deleted, pass "!H".
- By default, the function appends an "*" to the end of the string so that all handles other than those explicitly specified are cleaned up.

determineTableType function

Function used to determine what type of table is contained in the handle passed as an input parameter.

Location:

Parameters:

phBuffer INPUT HANDLE

Handle to either a temp-table object or a buffer object that contains the parameter values that you want to set.

Returns:

The return values are:

- **99** — Identifies a table that was previously formatted using one of the `setupTTFrom` functions.
 - **0** — Identifies a table that has no records in it. The 0 return value is also provided to indicate that there is no need to format the table using a `setupTTFrom` call.
 - An integer value corresponding to one of the table types. This value can be:
 - **1** — Stores parameters by position using the native data type.
 - **2** — Stores parameters by position using a character fields.
 - **3** — Stores parameters by name using native data type.
 - **4** — Stores parameters by name using a character field.
- NOTE:** For more information, see the “[calltables.i](#)” section.
- **?** — If the table type is not recognized

InvokeCall procedure

Procedure used to invoke a call.

Parameters:

`pcProc` INPUT CHARACTER

Name of the external or internal procedure, or function to be invoked.

`phInHandle` INPUT HANDLE

If `pcProc` is an internal procedure or function, this is the handle to the running persistent procedure that contains `pcProc`.

This parameter is optional. If nothing is specified, then `?` is passed.

`piType` INPUT INTEGER

Type of call being invoked. Must be either `PROCEDURE-CALL-TYPE` or `FUNCTION-CALL-TYPE`. These are constants that are recognized by the 4GL compiler.

phParamTable INPUT HANDLE

Handle to a temp-table previously created by a call to one of the setupTTFrom functions.

This parameter is optional. If nothing is specified, then ? is passed.

p1Persistent INPUT LOGICAL

Indicates whether the external procedure specified in pcProc should be invoked persistently.

phServer INPUT HANDLE

Handle to the AppServer where the call is to run.

This parameter is optional. If nothing is specified, then ? is passed.

p1Asynch INPUT LOGICAL

If set to **Yes**, the call is invoked asynchronously.

pcEventProc INPUT CHARACTER

Applies only if p1Asynch is set to **Yes**. This parameter identifies the procedure to be invoked when the asynchronous call completes.

This parameter is optional. If nothing is specified, then "" is passed.

phEventProcCtxte INPUT HANDLE

Contains the handle of the procedure that contains pcEventProc.

This parameter is optional. If nothing is specified, then ? is passed.

piCal OUTPUT INTEGER

Unique identifier to the ttCall record that is created for this call. You can use this call number until cleanupCall is run to call either obtainCallInfo or cleanupCall.

Notes:

- By default, all three handles are deleted when cleanupCall is invoked. To have all handles except the Call Handle deleted, pass "!H".
- By default, the function appends an "*" to the end of the string. As a result, all handles other than those explicitly specified are cleaned up.

obtainCallInfo function

Function used to obtain all the information about a call.

Parameters:

piCall INPUT INTEGER

A call number returned from a previous invokeCall request. This call number identifies the call you want to obtain.

PcReturnValue OUTPUT CHARACTER

The return value from the invoked call.

PhParamTable OUTPUT HANDLE

Handle to the temp-table that contains the parameters to the call.

Returns:

A handle to the DYNAMIC CALL object.

NOTE: If there is no call with the identifier specified, the unknown value ? is returned.

obtainParamPropValue function

Function that obtains a property value or parameter value from the session.

Parameters:

pcProperty INPUT CHARACTER

Name of a property whose value you want to obtain.

Returns:

A character string containing the value of the property.

NOTE: The function first tries to evaluate the property by calling getPropertyList. If this fails, the function attempts to obtain the value of the property by calling getSessionParam.

obtainProcHandle function

Function that obtains the handle to a procedure that was previously instantiated.

Parameters:

pcHandleName INPUT CHARACTER

Name of a handle to evaluate. The name can be one of the following:

- Name of a manager. If this is the case, getManagerHandle is called to determine the handle.
- Name of one of the predefined global variables that contain manager handles.
- Name of a procedure that could be running in the session.

Returns:

A handle to the persistent procedure.

NOTE: If the filename option is used and the filename is not found in the persistent procedure list, the unknown value ? is returned and the procedure is not instantiated.

setupTTFFromSig function

Function used to create a dynamic temp-table that invokes a call from the signature of a procedure or function.

Parameters:

pcTableName INPUT CHARACTER

Name to assign to the newly created temp-table.

phPersProc INPUT HANDLE

Handle to the persistent procedure that contains the call needed to derive the signature.

This parameter is optional. If nothing is specified, then ? is passed.

pcIntEntry INPUT CHARACTER

Procedure used to derive the signature for which a temp-table structure is to be created.

This parameter is optional. If nothing is specified, then ? is passed.

pcSignature INPUT CHARACTER

Signature of the procedure or function to invoke.

This parameter is optional. If nothing is specified, then "" or ? is passed.

phInitValueTT INPUT HANDLE

Handle to a temp-table that contains the parameter values to use for the call.

This parameter is optional. If nothing is specified, then ? is passed.

Returns:

A handle to the created temp-table.

Notes:

- You can specify phPersProc and pcIntEntry together, or you can specify pcSignature. If you specify all three, the contents of pcSignature is used.
- If you do not specify phInitValueTT, the values of the parameters are the default value for the data type of the parameter.
- The temp-table is created with initial values derived from the property values. No records are added to the temp-table. Creating a record in the table results in a record with the default values specified.

setupTTFFromString function

Function used to create a dynamics temp-table to invoke a call from a string containing the parameter definitions in the form *mode data type parameter*.

Parameters:

pcTableName INPUT CHARACTER

Name to assign the temp-table that you want to create.

pcRetValDT INPUT CHARACTER

Data type of the return value from the call. If left blank, the default is CHARACTER.

This parameter is optional. If nothing is specified, the "" is passed.

pcParamString INPUT CHARACTER

A string in the form *mode data type parameter*, *mode data type parameter*. The string contains the parameters you want to pass to the procedure or function that is being invoked. The string is a comma-separated list of parameters. Each parameter is in the form "*mode data type parameter*" where:

- *mode* is one of INPUT, OUTPUT, INPUT-OUTPUT, or OUTPUT-APPEND.
- *data type* is one of CHARACTER, DATE, LOGICAL, INTEGER, DECIMAL, RAW, HANDLE, TABLE-HANDLE, or ROWID
- *parameter* is the name of either a property that was previously set using `setPropertyList` or `setSessionParam`, or a single quoted constant that does not contain spaces or commas. If you specify a property, the procedure attempts to evaluate the property value by calling `getPropertyList`. If no property is available, or Dynamics is not running, a call is made to `getSessionParam` for the property value. If neither call succeeds, the unknown value is passed.

Returns:

A handle to the created temp-table.

Notes:

- When the temp-table is constructed, the function attempts to find a property using `getPropertyList` with the same name as the parameter. If none is found, a similar attempt is made to call `getSessionParam`. If no properties or parameters are found in context, the initial value is set to the unknown value.
- The temp-table is created with initial values that are derived from the property values. No records are added to the temp-table. A create of a record in the table results in a record with the default values specified.

setupTTFromTable function

Function used to create a dynamics temp-table to invoke a call.

Parameters:

pcTableName INPUT CHARACTER

Name to assign the temp-table you want to create.

pcRetValDT INPUT CHARACTER

Data type of the return value from the call. If left blank, the default is CHARACTER.

This parameter is optional. If nothing is specified, then "" is passed.

phParamTable INPUT HANDLE

Handle to a table that has the structure of either of the temp-tables in the in calltables.i file located in the adm2/ directory.

Returns:

Handle to the created temp-table.

Notes:

The temp-table is created with initial values derived from the records in the input table. No records are added to the temp-table. A create of a record in the table results in a record with the default values specified.

Index

A

Action properties

toolbar objects 6–38

actionCanRun (toolbar.p) 6–12

actionCaption (toolbar.p) 6–13

actionCategoryIsHidden (toolbar.p) 6–13

actionChecked (toolbar.p) 6–14

actionLabel (toolbar.p) 6–14

actionPublishCreate (toolbar.p) 6–14

actionTarget (toolbar.p) 6–15

actionTooltip (toolbar.p) 6–15

activeTarget (panel.p) 6–4

addColumnLink (wbtable.p) 9–2

addContextFields (webrep.p) 9–5

addDataObject (dynacontainer.w) 4–2

addDataTarget (sbo.p) 4–25

addLink (smart.p) 2–2

addMessage (smart.p) 2–4

AddMode property 9–7

addNavigationSource (sbo.p) 4–26

addNode (treeview.p) 3–72

addNotFoundMessage (query.p) 5–2

addQueryWhere (query.p) 5–2

addQueryWhere (sbo.p) 4–26

addRecord (browser.p) 3–45

addRecord (viewer.p) 3–62

addRecord datavis.p) 3–22

addRow (data.p) 5–27

addRow (sbo.p) 4–27

addSearchCriteria (webrep.p) 9–5

addTDModifier (wbtable.p) 9–6

adjustTabOrder (smart.p) 2–5

- admweb.p
 - destroy 9–20
 - destroyObject 9–21
 - getAttribute 9–29
 - getWebState 9–35
 - getWebTimeout 9–36
 - getWebTimeRemaining 9–36
 - getWebToHdlr 9–36
 - set-attribute-list 9–47
 - setWebState 9–57
 - setWebToHdlr 9–57
 - timingOut 9–58
- AllFieldHandles property 3–80
- AllFieldNames property 3–80
- anyKey (combo.p) 7–13
- anyKey (lookup.p) 7–22
- anyKey (select.p) 7–3
- anyMessage (smart.p) 2–6
- anyMessage (webrep.p) 9–6
- appendContainedObjects (sbo.p) 4–27
- appserver.p
 - disconnectObject 2–31
- ApplyActionOnExit property 3–80
- applyCellEntry (browser.p) 3–46
- applyContextFromServer (containr.p) 4–2
- applyEntry (browser.p) 3–46
- applyEntry (smart.p) 2–6
- ApplyExitOnAction property 3–81
- applyFilter (filter.p) 3–36
- applyLayout (visual.p) 3–2
- appserver.p
 - bindServer 2–30
 - destroyObject 2–31
 - destroyServerObject 2–31
 - initializeServerObject 2–31
 - runServerObject 2–32
 - runServerProcedure 2–32
 - unbindServer 2–32
- AppService property 2–34
- ASBound property 2–34
- ASDivision property 2–34
- ASHandle property 2–35
- ASHasStarted property 2–35
- ASInitializeOnRun property 2–35
- askQuestion (exportdata.p) 5–27
- assignColumnFormat (webrep.p) 9–7
- assignColumnHelp (webrep.p) 9–8
- assignColumnLabel (webrep.p) 9–8
- assignColumnWidth (webrep.p) 9–9
- assignContainedProperties (containr.p) 4–3
- assignCurrentMappedObject (sbo.p) 4–28
- assignDBRow (query.p) 5–3
- assignExtentAttribute (webrep.p) 9–9
- assignFields (html-map.p) 9–10
- assignFields (wbdata.p) 9–10
- assignFocusedWidget (visual.p) 3–2
- assignLinkProperty (smart.p) 2–7
- AssignList property 5–74
- assignMaxGuess (browser.p) 3–47
- assignMaxGuess (sbo.p) 4–28

assignNodeValue (xml.p) 8–9
assignPageProperty (containr.p) 4–4
assignQuerySelection (query.p) 5–4
assignQuerySelection (sbo.p) 4–28
assignTDM Modifier (wbtable.p) 9–11
assignUnsubscribe (consumer.p) 8–3
assignWidgetValue (visual.p) 3–3
assignWidgetValueList (visual.p) 3–4
Attribute property 9–12
AutoCommit property 5–74
AutoSort property 3–81
AvailMenuActions property 6–30
AvailToolbarActions property 6–30
AvailToolbarBands property 6–30

B

b2b.p
 callOutParams 8–16
 characterValue 8–17
 createSchemaAttributes 8–17
 createSchemaChildren 8–17
 createSchemaPath 8–18
 dataSource 8–18
 defineMapping 8–18
 destroyObject 8–19
 endDocument 8–19
 endElement 8–19
 findDataRow 8–20
 initializeObject 8–20
 loadProducerSchema 8–20
 loadSchema 8–21
 mapNode 8–21
 NotFoundError 8–22
 numParameters 8–22

 processMappings 8–22
 processMessages 8–23
 produceAttributes 8–23
 produceChildren 8–24
 produceDocument 8–24
 receiveHandler 8–24
 rowNotFoundError 8–25
 schemaField 8–25
 sendHandler 8–26
 sendMessage 8–26
 startDataRow 8–26
 startElement 8–27
 storeNodeValue 8–27
 storeParameterNode 8–28
 storeParameterValue 8–28
 targetProcedure 8–29

BaseQueryString property 7–33
batchRowAvailable (data.p) 5–28
batchServices (data.p) 5–5
beginTransactionValidate (data.p) 5–28
bindServer (appserver.p) 2–30
blankField (filter.p) 3–37
blankFields (filter.p) 3–37
blankFillIn (filter.p) 3–38
blankWidget (visual.p) 3–5
BlockDataAvailable property 4–53
BoxRectangle property 6–31
BoxRectangle2 property 6–31
BrowseFieldFormats property 7–33
BrowseFields property 7–33
BrowseHandle property 3–81
browseHandler (select.p) 7–3

browser.p
 addRecord 3–45
 applyCellEntry 3–46
 applyEntry 3–46
 assignMaxGue 3–47
 calcWidth 3–47
 cancelNew 3–47
 cancelRecord 3–48
 colValues 3–48
 copyRecord 3–49
 dataAvailable 3–49
 defaultAction 3–50
 deleteComplete 3–50
 deleteRecord 3–50
 destroyObject 3–51
 disableFields 3–51
 displayFields 3–52
 enableFields 3–52
 fetchDataSet 3–53
 filterActive 3–53
 hasActiveAudit 3–53
 hasActiveComments 3–54
 initializeObject 3–54
 launchFolderWindow 3–55
 offEnd 3–55
 offHome 3–55
 onEnd 3–55
 onHome 3–56
 refreshBrowse 3–56
 resetRecord 3–56
 resizeBrowse 3–57
 resizeObject 3–57
 rowChanged 3–58
 rowDisplay 3–58
 rowVisible 3–58
 searchTrigger 3–59
 startSearch 3–59
 stripCalcs 3–59
 toolbar 3–60
 updateRecord 3–60
 updateState 3–61
 updateTitle 3–61
 ValueChanged 3–61
 viewObject 3–62

BrowseTitle property 7–33

bufferCopyDBToRO (query.p) 5–6

bufferHandle (html-map.p) 9–12

bufferHandle (webrep.p) 9–13

buildAllMenus (toolbar.p) 6–16

buildList (select.p) 7–3

BuildSequence property 7–34

ButtonCount property 6–31

C

cache.p
 findCacheItem 4–13
 registerCacheItem 4–6, 4–20

CacheDuration property 5–74

CalculatedColumns property 5–74

calcWidth (browser.p) 3–47

CalcWidth property 3–81

CallerObject property 4–54

CallerProcedure property 4–54

CallerWindow property 4–54

callOutParams (b2b.p) 8–16

callstring.p procedure A–26

callstringtt.p A–5

callstringtt.p procedure A–28

calltable.p A–7

calltable.p procedure A–31

calltables.i A–11

calltablett.p A–10

calltablett.p procedure A–33

callttparam.i A–12

- CancelBrowseOnExit property 7–34
- cancelNew (browser.p) 3–47
- cancelNew (sbo.p) 4–29
- cancelObject (containr.p) 4–5
- cancelObject (datavis.p) 3–22
- cancelRecord (browser.p) 3–48
- cancelRecord (datavis.p) 3–23
- cancelRecord (viewer.p) 3–63
- cancelRow (data.p) 5–29
- cancelRow (sbo.p) 4–29
- canFindAction (toolbar.p) 6–2
- canFindCategory (toolbar.p) 6–2
- canNavigate (data.p) 5–29
- canNavigate (datavis.p) 3–24
- canNavigatre (sbo.p) 4–30
- CascadeOnBrowse property 4–54
- categoryActions (toolbar.p) 6–16
- categoryLink (toolbar.p) 6–2
- changeCursor (smart.p) 2–8
- ChangedEvent property 7–34
- changePage (containr.p) 4–5
- characterValue (b2b.p) 8–17
- CheckCurrentChanged property 5–74
- CheckLastOnOpen property 5–75
- checkRule (toolbar.p) 6–3
- ChildDataKey property 2–35
- cleanupCall procedure A–36
- ClientID property 8–33
- ClientProxyHandle property 5–75
- ClientRect property 3–81
- clientSendRows (data.p) 5–30
- closeQuery (data.p) 5–31
- closeQuery (query.p) 5–7
- Col property 3–81
- collectChanges (datavis.p) 3–24
- colStringValue (data.p) 5–31
- Column properties
 - container objects 3–106, 4–66
 - field objects 7–50
 - query objects 5–96
- columnDataType (webrep.p) 9–13
- columnFormat (webrep.p) 9–13
- columnHandle (webrep.p) 9–14
- columnHelp (webrep.p) 9–14
- columnHTMLName (html-map.p) 9–14
- columnHTMLName (webrep.p) 9–15
- columnLabel (webrep.p) 9–16
- ColumnPhysicalColumn (query.p) 5–6
- ColumnPhysicalTable (query.p) 5–7
- columnReadOnly (webrep.p) 9–16
- columnStringValue (webrep.p) 9–16
- columnTable (html-map.p) 9–17
- columnTDModifier (wbtable.p) 9–17
- columnValMsg (wbdata.p) 9–18
- colValues (browser.p) 3–48

colValues (data.p) 5–32

colValues (query.p) 5–8

colValues (sbo.p) 4–30

combo.p

anyKey 7–13

createDataSource 7–13

createLabel 7–13

destroyCombo 7–14

destroyObject 7–14

disable_UI 7–14

disableField 7–14

displayCombo 7–15

displayField 7–15

enableField 7–15

endMove 7–16

enterCombo 7–16

hideObject 7–16

initializeCombo 7–16

insertExpression 7–17

leaveCombo 7–17

newQueryString 7–18

newWhereClause 7–19

prepareField 7–19

refreshChildDependencies 7–20

refreshCombo 7–20

resizeObject 7–20

returnParentFieldValues 7–21

showQuery 7–21

valueChanged 7–21

viewObject 7–22

whereClauseBuffer 7–22

ComboBuffer property 7–34

ComboDelimiter property 7–34

ComboFlag property 7–35

ComboFlagValue property 7–35

ComboHandle property 7–35

ComboQuery property 7–36

ComboSort property 7–36

Commit (data.p) 5–32

commitData (data.p) 5–33

CommitSource property 4–54

CommitSourceEvents property 4–55

CommitTarget property 4–55

CommitTargetEvents property 4–55

commitTransaction (data.p) 5–33

commitTransaction (sbo.p) 4–31

confirmCancel (containr.p) 4–10

confirmCancel (datavis.p) 3–25

confirmCommit (datavis.p) 3–25

confirmCommit (query.p) 5–8

confirmContinue (datavis.p) 3–26

confirmContinue (query.p) 5–8

confirmContinue (sbo.p) 4–31

confirmDelete (datavis.p) 3–26

confirmExit (containr.p) 4–7

confirmExit (datavis.p) 3–26

confirmOk (containr.p) 4–7

confirmOk (datavis.p) 3–27

confirmUndo (datavis.p) 3–27

confirmUndo (query.p) 5–9

constructMenu (toolbar.p) 6–16

constructObject (containr.p) 4–8

constructObject (html-map.p) 9–18

constructToolbar (toolbar.p) 6–16

- consumer.p
 - assignUnsubscribe 8–3
 - createConsumers 8–3
 - defineDestination 8–4
 - destroyObject 8–4
 - errorHandler 8–4
 - initializeObject 8–5
 - messageHandler 8–5
 - processDestinations 8–5
 - startWaitFor 8–6
 - stopHandler 8–6
- ConsumerSchema property 8–33
- ContainedDataColumns property 4–55
- ContainedDataObjects property 4–56
- containedProperties (sbo.p) 4–8
- Container objects
 - column properties 3–106, 4–66
- ContainerHandle property 2–36
- ContainerHidden property 2–36
- ContainerSource property 2–36
- ContainerSourceEvents property 2–36, 4–56
- ContainerTarget property 4–56
- ContainerTargetEvents property 4–56
- ContainerType property 2–37
- containr.p
 - assignContainedProperties 4–3
 - assignPageProperty 4–4
 - cancelObject 4–5
 - changePage 4–5
 - confirmCancel 4–10
 - confirmExit 4–7
 - confirmOk 4–7
 - constructObject 4–8
 - ContextandDestroy 4–9
 - createObjects 4–9
 - deletePage 4–11
 - destroyObject 4–11
 - disablePageInFolder 4–12
 - enablePagesInFolder 4–12
 - fetchContainedData 4–12
 - hidePage 4–13
 - initializeObject 4–14
 - initializeVisualContainer 4–14
 - initPages 4–14
 - IsUpdateActive 4–15
 - notifyPage 4–16
 - obtainContextForServer 4–17
 - okObject 4–18
 - pageNTargets 4–18
 - passThrough 4–19
 - removePageNTarget 4–20
 - resizeWindow 4–21
 - selectPage 4–21
 - targetPage 4–22
 - toolbar 4–22
 - updateActive 4–23
 - viewObject 4–24
 - viewPage 4–24
- ContextandDestroy (containr.p) 4–9
- ContextAndInitialize property 4–56
- ContextForServer property 8–33
- copyRecord (browser.p) 3–49
- copyRecord (datavis.p) 3–27
- copyRecord (viewer.p) 3–64
- copyRow (data.p) 5–33
- copyRow (sbo.p) 4–32
- cotainr.p
 - applyContextFromServer 4–2
- countButtons (panel.p) 6–5
- createAttribute (xml.p) 8–9
- createConsumers (consumer.p) 8–3
- createControls (smart.p) 2–8
- createData (data.p) 5–34
- createDataSource (combo.p) 7–13

createDocument (router.p) 8–29

createElement (xml.p) 8–10

createField (filter.p) 3–38

CreateHandles property 3–82

createLabel (combo.p) 7–13

createLabel (filter.p) 3–39

createLabel (lookup.p) 7–23

createLabel (select.p) 7–4

createMenuTable (toolbar.p) 6–17

createMenuTable2 (toolbar.p) 6–18

createNode (xml.p) 8–10

createObjects (containr.p) 4–9

createObjects (data.p) 5–34

createOperator (filter.p) 3–40

createRow (data.p) 5–34

createSchemaAttributes (b2b.p) 8–17

createSchemaChildren (b2b.p) 8–17

createSchemaPath (b2b.p) 8–18

createText (xml.p) 8–11

createToolba (toolbar.p) 6–18

CtrlFrameHandle 3–82

CurrentDescValue property 7–36

CurrentKeyValue property 7–36

currentMappedObject (sbo.p) 4–32

CurrentMessage property 8–34

CurrentMessageId property 8–34

CurrentPage property 4–56

CurrentRowModified property 5–75

D

data.p

addRow 5–27

batchRowAvailable 5–28

batchServices 5–5

beginTransactionValidate 5–28

cancelRow 5–29

canNavigate 5–29

clientSendRows 5–30

closeQuery 5–31

colStringValue 5–31

colValues 5–32

Commit 5–32

commitData 5–33

commitTransaction 5–33

copyRow 5–33

createData 5–34

createObjects 5–34

createRow 5–34

dataAvailable 5–35

dataContainerHandle 5–36

datupdateManualForeignFields 5–71

deleteData 5–36

deleteRow 5–37

describeSchema 5–37

destroyObject 5–37

destroyServerObject 5–38

doBuildUpd 5–38

doCreateUpdate 5–39

doEmptyTempTable 5–39

doReturnUpd 5–40

doUndoDelete 5–40

doUndoRow 5–41

doUndoTrans 5–41

doUndoUpdate 5–41

endClientDataRequest 5–41

endTransactionValidate 5–42

exportData 5–42

fetchBatch 5–43

fetchFirst 5–43

fetchLast 5–44

fetchNext 5–44

fetchPrev 5–44

fetchRow 5–45

fetchRowIdent 5–45

findRow 5–46
findRowWhere 5–47
firstRowIds 5–48
hasActiveAudit 5–48
hasActiveComments 5–48
hasForeignKeyChanged 5–48
hasOneToOneTarget 5–49
initializeLogicObject 5–49
initializeObject 5–49
initializeServerObject 5–50
isUpdateActive 5–50
isUpdatePending 5–50
linkState 5–51
linkStateHandle 5–51
newRowObject 5–52
obtainContextForServer 5–52
openDataQuery 5–52
openQuery 5–53
postTransactionValidate 5–53
prepareErrorsForReturn 5–53
preTransactionValidate 5–54
printToCrystal 5–54
pushTableAndValidate 5–54
refreshRow 5–55
remoteCommit 5–55
remoteSendRow 5–56
repositionRowObject 5–57
resortQuery 5–58
rowAvailable 5–58
rowValues 5–60
saveContextAndDestroy 5–61
sendRows 5–62
serverCommit 5–63
serverSendRows 5–64
startServerObject 5–65
submitCommit 5–65
submitForeignKey 5–66
submitRow 5–67
synchronizeProperties 5–68
tableOut 5–68
transferToExcel 5–69
undoClientUpdate 5–70
undoTransaction 5–70
updateAddQueryWhere 5–70
updateData 5–71
updateQueryPosition 5–72
updateRow 5–72
updateState 5–73

dataAvailable (browser.p) 3–49
dataAvailable (data.p) 5–35
dataAvailable (datavis.p) 3–28
dataAvailable (filter.p) 3–40
dataAvailable (html-map.p) 9–19
dataAvailable (query.p) 5–10
dataAvailable (sbo.p) 4–32
dataAvailable (select.p) 7–4
DataColumns ByTable (queryext.p) 5–76
DataColumns property 4–57, 5–76
dataContainerHandle (data.p) 5–36
DataDelimiter property 5–76
DataFieldDefs property 5–76
DataHandle property 4–57, 5–77
DataIsFetched property 5–77
DataLinksEnabled property 2–37
datalogic.i
 rowObjectValidate 5–59
DataLogicObject property 5–77
DataLogicProcedure property 5–78
DataModified property 3–83, 5–77, 7–37
DataObject property 3–83
dataObjectHandle (sbo.p) 4–33
DataObjectName property 4–57
DataObjectOrdering property 4–57
DataQueryBrowsed property 4–57, 5–78
DataQueryString property 5–78

- DataReadBuffer property 5–78
- DataReadColumns property 5–78
- DataReadHandler property 5–79
- DataSignature property 3–83, 5–79
- dataSource (b2b.p) 8–18
- DataSource property 2–37
- DataSourceEvents property 2–37
- DataSourceFilter property 7–37
- DataSourceName property 7–37
- DataSourceNames property 2–37
- DataTarget property 2–38
- DataTargetEvents property 2–38
- dataValue
 - filter.p 3–41
- dataValue (filter.p) 3–41
- DataValue property 7–37
- datavis.p
 - addRecord 3–22
 - cancelObject 3–22
 - cancelRecord 3–23
 - canNavigate 3–24
 - collectChanges 3–24
 - confirmCancel 3–25
 - confirmCommit 3–25
 - confirmContinue 3–26
 - confirmDelete 3–26
 - confirmExit 3–26
 - confirmOk 3–27
 - confirmUndo 3–27
 - copyRecord 3–27
 - dataAvailable 3–28
 - deleteRecord 3–28
 - disableFields 3–29
 - displayObjects 3–29
 - displayRecord 3–30
 - enableFields 3–30
 - initializeObject 3–30
 - IsUpdateActive 3–31
 - linkStateHandler 3–31
 - okObject 3–31
 - okToContinueProcedure 3–32
 - queryPosition 3–32
 - resetRecord 3–33
 - showDataMessages 3–33
 - showDataMessagesProcedure 3–34
 - updateMode 3–34
 - updateRecord 3–35
 - updateState 3–35
 - validateFields 3–35
- datupdateManualForeignFields (data.p) 5–71
- DBAware property 2–38
- dbColumnName (query.p) 5–11
- dbColumnHandle (query.p) 5–11
- DbNames property 5–79
- defaultAction (browser.p) 3–50
- DefaultCharWidth property 3–83
- DefaultEditorLines property 3–84
- DefaultLayout property 3–84
- DefaultWidth property 3–84
- defineAction (toolbar.p) 6–3
- DefineAnyKeyTrigger property 7–37
- defineDataObject (query.p) 5–12
- defineDestination (consumer.p) 8–4
- defineMapping (b2b.p) 8–18
- deleteBuffer (wbdata.p) 9–19
- deleteComplete (browser.p) 3–50
- deleteComplete (sbo.p) 4–33
- deleteData (data.p) 5–36

- deleteDocument (xml.p) 8–11
- deleteNode (treeview.p) 3–73
- deleteObjects (filter.p) 3–41
- deleteOffsets (html-map.p) 9–20
- deletePage (containr.p) 4–11
- deleteRecord (browser.p) 3–50
- deleteRecord (datavis.p) 3–28
- deleteRecordStatic (query.i) 5–13
- deleteRow (data.p) 5–37
- deleteRow (sbo.p) 4–33
- deleteRow (wbdata.p) 9–20
- deleteTree (treeview.p) 3–74
- describeSchema (data.p) 5–37
- DescSubstitute property 7–38
- Destination property 8–34
- DestinationList property 8–34
- Destinations property 8–34
- destroy (admweb.p) 9–20
- destroyCombo (combo.p) 7–14
- destroyDataObject (webrep.p) 9–21
- destroyLookup (lookup.p) 7–23
- destroyObject (admweb.p) 9–21
- destroyObject (appserver.p) 2–31
- destroyObject (b2b.p) 8–19
- destroyObject (browser.p) 3–51
- destroyObject (combo.p) 7–14
- destroyObject (consumer.p) 8–4
- destroyObject (containr.p) 4–11
- destroyObject (data.p) 5–37
- destroyObject (lookup.p) 7–23
- destroyObject (messaging.p) 8–2
- destroyObject (producer.p) 8–6
- destroyObject (select.p) 7–5
- destroyObject (smart.p) 2–9
- destroyObject (webrep.p) 9–21
- destroyObject (xml.p) 8–11
- destroySelection (select.p) 7–5
- destroyServerObject (appserver.p) 2–31
- destroyServerObject (data.p) 5–38
- destroyServerObject (sbo.p) 4–33
- DestroyStateless property 5–80
- determineTableType function A–36
- DirectionList property 8–35
- disable_UI (combo.p) 7–14
- disable_UI (lookup.p) 7–24
- disable_UI (select.p) 7–6
- disableActions (panel.p) 6–5
- disableButton (select.p) 7–5
- DisabledActions property 6–32
- disableField (combo.p) 7–14
- disableField (lookup.p) 7–24
- disableField (select.p) 7–5
- disableFields (browser.p) 3–51
- disableFields (datavis.p) 3–29

- disableFields (filter.p) 3–41
- disableFields (viewer.p) 3–65
- disableObject (treeview.p) 3–74
- disableObject (visual.p) 3–6
- DisableOnInit property 3–84
- disablePageInFolder (containr.p) 4–12
- disableRadioButton (visual.p) 3–6
- disableWidget (visual.p) 3–7
- DisconnectAppServer property 5–80
- disconnectObject (appserver.p) 2–31
- disconnectObject (webrep.p) 9–21
- dispatchUtilityProc (html-map.p) 9–22
- displayActions (toolbar.p) 6–4
- displayCombo (combo.p) 7–15
- DisplayDataType property 7–38
- DisplayedField property 3–84
- DisplayedTables property 3–85
- displayField (combo.p) 7–15
- displayField (lookup.p) 7–24
- DisplayField property 7–38
- displayFields (browser.p) 3–52
- displayFields (html-map.p) 9–22
- displayFields (viewer.p) 3–66
- DisplayFieldsSecurity property 3–84
- DisplayFormat property 7–38
- displayLinks (smart.p) 2–10
- displayLookup (lookup.p) 7–24
- displayObjects (datavis.p) 3–29
- displayRecord (datavis.p) 3–30
- DisplayValue property 7–38
- doBuildUpd (data.p) 5–38
- doCreateUpdate (data.p) 5–39
- DocumentElement property 8–35
- DocumentHandle property 8–35
- DocumentInitialized property 8–35
- doEmptyTempTable (data.p) 5–39
- Domain property 8–35
- doReturnUpd (data.p) 5–40
- doUndoDelete (data.p) 5–40
- doUndoRow (data.p) 5–41
- doUndoTrans (data.p) 5–41
- doUndoUpdate (data.p) 5–41
- Down property 3–85
- DTDPublicIdList property 8–36
- DTDSYSTEMID property 8–36
- DTDSYSTEMIDList property 8–36
- dynacontainer.w
 - addDataObject 4–2
- DynamicSDOPROCEDURE property 4–58
- dynlaunch.i A–2
 - AppServer calls A–2
 - file arguments A–2
 - include file A–2
 - when to use A–2

E

- EdgePixels property 6–32, 7–39
- Editable property 3–85
- editInstanceProperties (smart.p) 2–10
- emptyTree (treeview.p) 3–74
- enableActions (panel.p) 6–6
- enableButton (lookup.p) 7–25
- enableButton (select.p) 7–6
- enabledField (lookup.p) 7–25
- EnabledFields property 3–86
- EnabledHandles property 3–86
- EnabledObjFlds property 3–85
- EnabledObjFldsToDisable property 3–85, 5–80
- EnabledObjHdls property 3–86
- EnabledTables property 5–81
- enableField (combo.p) 7–15
- enableField (select.p) 7–6
- EnableField property 7–39
- enableFields (browser.p) 3–52
- enableFields (datavis.p) 3–30
- enableFields (filter.p) 3–41
- enableFields (html-map.p) 9–23
- enableFields (viewer.p) 3–68
- enableObject (panel.p) 6–6
- enableObject (treeview.p) 3–74
- enableObject (visual.p) 3–8
- EnableOnAdd property 7–39
- enablePagesInFolder (containr.p) 4–12
- enableRadioButton (visual.p) 3–8
- enableWidget (visual.p) 3–9
- endClientDataRequest (data.p) 5–41
- endClientDataRequest (sbo.p) 4–34
- endDocument (b2b.p) 8–19
- endElement (b2b.p) 8–19
- endMove (combo.p) 7–16
- endMove (lookup.p) 7–25
- endMove (select.p) 7–6
- endTransactionValidate (data.p) 5–42
- enterCombo (combo.p) 7–16
- enterLookup (lookup.p) 7–25
- enterSelect (select.p) 7–7
- errorHandler (consumer.p) 8–4
- errorHandler (messaging.p) 8–2
- exclusiveLockBuffer (wbdata.p) 9–24
- ExitBrowseOnAction property 7–39
- exitObject (smart.p) 2–11
- ExpandOnAdd property 3–86
- exportData (data.p) 5–42
- exportdata.p
 - askQuestion 5–27
- extentAttribute (webrep.p) 9–24
- extentValue (webrep.p) 9–25
- ExternalRefList property 8–36

F

- fetchBatch (data.p) 5–43
- fetchBatch (sbo.p) 4–34
- fetchContainedData (containr.p) 4–12
- fetchContainedData (sbo.p) 4–34
- fetchContainedRows (sbo.p) 4–35
- fetchCurrent (webrep.p) 9–25
- fetchCurrentBatch (query.p) 5–15
- fetchDataSet (browser.p) 3–53
- fetchDOProperties (sbo.p) 4–35
- fetchFirst (data.p) 5–43
- fetchFirst (query.p) 5–13
- fetchFirst (sbo.p) 4–35
- fetchFirst (webrep.p) 9–25
- fetchFirstBatch (query.p) 5–14
- fetchLast (data.p) 5–44
- fetchLast (query.p) 5–13
- fetchLast (sbo.p) 4–36
- fetchLast (whtable.p) 9–26
- fetchLast (webrep.p) 9–26
- fetchLastBatch (query.p) 5–15
- fetchMessages (smart.p) 2–12
- fetchNext (data.p) 5–44
- fetchNext (query.p) 5–14
- fetchNext (sbo.p) 4–36
- fetchNext (whtable.p) 9–26
- fetchNext (webrep.p) 9–27
- fetchNextBatch (query.p) 5–14
- FetchOnOpen property 4–58
- fetchPrev (data.p) 5–44
- fetchPrev (query.p) 5–14
- fetchPrev (sbo.p) 4–36
- fetchPrev (whtable.p) 9–27
- fetchPrev (webrep.p) 9–27
- fetchPrevBatch (query.p) 5–14
- fetchRow (data.p) 5–45
- fetchRowIdent (data.p) 5–45
- Field objects
 - column properties 7–50
- field.p
 - initializeObject 7–2
 - resizeObject 7–2
- FieldColumn property 3–87
- FieldEnabled property 7–39
- fieldExpression (webrep.p) 9–28
- FieldFormats property 3–87
- FieldHandles property 3–87
- FieldHelpIds property 3–87
- FieldHidden property 7–40
- FieldLabel property 7–40
- FieldLabels property 3–88
- fieldLookup (filter.p) 3–42
- fieldModified (viewer.p) 3–68
- FieldName property 7–40
- FieldOperatorStyles property 3–88

- fieldProperty (filter.p) 3–42
- FieldsEnabled property 3–87
- FieldToolTip property 7–40
- FieldToolTips property 3–88
- FieldWidths property 3–88
- FillBatchOnRepos property 5–81
- filter.p
 - applyFilter.p 3–36
 - blankField 3–37
 - blankFields 3–37
 - blankFillIn 3–38
 - createField 3–38
 - createLabel 3–39
 - createOperator 3–40
 - dataAvailable 3–40
 - deleteObjects 3–41
 - disableFields 3–41
 - enableFields 3–41
 - fieldLookup 3–42
 - fieldProperty 3–42
 - initializeObject 3–42
 - insertFieldProperty 3–43
 - removeSpace 3–43
 - resetFields 3–44
 - showDataMessages 3–44
 - unBlankFillin 3–44
 - unBlankLogical 3–45
- filterActive (browser.p) 3–53
- FilterActive property 3–88, 5–82
- FilterAvailable property 5–82
- filterContainerHandler (query.p) 5–15
- FilterSource property 4–58
- filterState (toolbar.p) 6–19
- FilterTarget property 3–89
- FilterTargetEvents property 3–89
- FilterWindow property 5–81
- findCacheItem (cache.p) 4–13
- findDataRow (b2b.p) 8–20
- findRecords (html-map.p) 9–28
- findRow (data.p) 5–46
- findRowWhere (data.p) 5–47
- findRowWhere (sbo.p) 4–37
- firstBufferName (query.p) 5–16
- FirstResultRow property 5–81
- firstRowIds (data.p) 5–48
- firstRowIds (query.p) 5–16
- FirstRowNum property 5–81
- fixQueryString (smart.p) 2–13
- FlagValue property 7–40
- FolderWindowToLaunch property 3–89
- ForeignFields property 5–82
- ForeignValues property 5–82
- Format property 7–40
- formattedValue (select.p) 7–7
- formattedWidgetValue (visual.p) 3–10
- formattedWidgetValueList (visual.p) 3–10
- FrameMinHeightChars property 3–89
- FrameMinWidthChars property 3–90
- FullRowSelect property 3–90

G

- getAttribute
 - admweb.p 9–29
- getContextFields
 - webrep.p 9–29
- getCurrentPage
 - html-map.p 9–30
- getCurrentRowids (webrep.p) 9–30
- getDeleteTables
 - wbdata.p 9–30
- getForeignFieldList (webrep.p) 9–31
- getFrameHandle (wbdata.p) 9–31
- getNavigationMode (webprep.p) 9–31
- getNextHtmlField (html-map.p) 9–32
- getQueryEmpty (webrep.p) 9–33
- getQueryWhere (webrep.p) 9–33
- getRowids (webrep.p) 9–33
- getSearchColumns (webrep.p) 9–33
- getServerConnection (webrep.p) 9–34
- getTableRowids (webrep.p) 9–34
- getTableRows (wbtable.p) 9–34
- getTables (html-map.p) 9–34
- getTables (webrep.p) 9–35
- getUpdateMode (webrep.p) 9–35
- getWebState (admweb.p) 9–35
- getWebTimeout (admweb.p) 9–36
- getWebTimeRemaining (admweb.p) 9–36
- getWebToHdlr (admweb.p) 9–36
- GroupAssignHidden property 3–90

- GroupAssignSource property 3–90, 5–82
- GroupAssignSourceEvents property 3–91, 5–83
- GroupAssignTarget property 3–91, 5–83
- GroupAssignTargetEvents property 3–91, 5–83

H

- hasActiveAudit (browser.p) 3–53
- hasActiveAudit (data.p) 5–48
- hasActiveComments (browser.p) 3–54
- hasActiveComments (data.p) 5–48
- hasActiveGaTarget (panel.p) 6–6
- hasForeignKeyChanged (data.p) 5–48
- hasOneToOneTarget (data.p) 5–49
- Height property 3–91
- HelpId property 7–41
- HiddenActions property 6–32
- HiddenMenuBands property 6–32
- HiddenToolbarBands property 6–32
- hideObject (combo.p) 7–16
- hideObject (lookup.p) 7–26
- hideObject (select.p) 7–7
- hideObject (smart.p) 2–16
- hideObject (toolbar.p) 6–19
- HideOnInit property 3–91
- hidePage (containr.p) 4–13
- HideSelection property 3–91

- hideWidget (visual.p) 3–12
- highlightWidget (visual.p) 3–12
- htmAssociate (html-map.p) 9–37
- HTMLAlert (webrep.p) 9–37
- HTMLColumn (wbtable.p) 9–38
- html-map.p
 - assignField 9–10
 - bufferHandle 9–12
 - columnHTMLName 9–14
 - columnTable 9–17
 - constructObject 9–18
 - dataAvailable 9–19
 - deleteOffsets 9–20
 - dispatchUtilityProc 9–22
 - displayFields 9–22
 - enableFields 9–23
 - findRecords 9–28
 - getCurrentPage 9–30
 - getNextHtmlField 9–32
 - getTables 9–34
 - htmAssociate 9–37
 - initializeObject 9–39
 - inputFields 9–39
 - outputFields 9–42
 - readOffsets 9–45
- HTMLSetFocus (wbrep.p) 9–38
- HTMLTable (wbtable.p) 9–39
- I**
- ILComHandle property 3–92
- Image property 6–33
- ImageHeight property 3–92
- imageName (toolbar.p) 6–19
- ImagePath property 6–33
- ImageWidth property 3–92
- inactiveLinks property 2–38
- Indentation property 3–92
- indexInformation (query.p) 5–17
- IndexInformation property 5–83
- initAction (toolbar.p) 6–4
- initDataObjectOrdering (sbo.i) 4–36
- initializeBrowse 7–8
- initializeBrowse (lookup.p) 7–26
- initializeCombo (combo.p) 7–16
- initializeLogicObject (data.p) 5–49
- initializeLookup (lookup.p) 7–26
- initializeObject (b2b.p) 8–20
- initializeObject (browser.p) 3–54
- initializeObject (consumer.p) 8–5
- initializeObject (containr.p) 4–14
- initializeObject (data.) 5–49
- initializeObject (datavis.p) 3–30
- initializeObject (field.p) 7–2
- initializeObject (filter.p) 3–42
- initializeObject (html-map.p) 9–39
- initializeObject (messaging.p) 8–2
- initializeObject (panel.p) 6–7
- initializeObject (producer.p) 8–7
- initializeObject (query.p) 5–17
- initializeObject (router.p) 8–30
- initializeObject (sbo.p) 4–38
- initializeObject (select.p) 7–8
- initializeObject (smart.p) 2–13

initializeObject (toolbar.p) 6–4, 6–20
initializeObject (tvcontroller.p) 4–52
initializeObject (viewer.p) 3–69
initializeObject (visual.p) 3–14
initializeSelection (select.p) 7–8
initializeServerObject (appserver.p) 2–31
initializeServerObject (data.p) 5–50
initializeServerObject (sbo.p) 4–38
initializeVisualContainer (containr.p) 4–14
InitialPageList property 4–58
initPages (containr.p) 4–14
initProp (query.i) 5–17
InMessageSource property 8–36
InMessageTarget property 4–58
InnerLines property 7–41
inputFields (html-map.p) 9–39
insertExpression (combo.p) 7–17
insertExpression (lookup.p) 7–27
insertExpression (query.p) 5–18
insertFieldProperty (filter.p) 3–43
insertMenu (toolbar.p) 6–20
instanceOf (smart.p) 2–14
InstanceProperties property 2–38
instancePropertyList (smart.p) 2–15
InternalEntries property 5–83
InternalRefList property 8–37
internalSchemaFile (router.p) 8–30

InvokeCall procedure A–37
isNodeExpanded (treeview.p) 3–75
IsUpdateActive (containr.p) 4–15
isUpdateActive (data.p) 5–50
IsUpdateActive (datavis.p) 3–31
isUpdatePending (data.p) 5–50
isUpdatePending (sbo.p) 4–38

J

JMSpartition property 8–37
JMSPassword property 8–37
JMSsession property 8–37
JMSuser property 8–38
joinExternalTables (webrep.p) 9–40
joinForeignFields (webrep.p) 9–41

K

KeepChildPositions property 3–92
KeyDataType property 7–41
KeyField property 7–41
KeyFields property 5–84, 7–41
KeyFormat property 7–41

L

Label property 7–41
LabelEdit property 3–93
LabelHandle property 7–42
LastCommitErrorKeys property 4–59, 5–84

- LastCommitErrorType 4–59, 5–84
- LastCommitErrorType property 5–85
- LastDbRowIdent property 5–85
- LastResultRow property 5–86
- LastRowNum property 5–86
- launchFolderWindow (browser.p) 3–55
- LayoutOptions property 3–93
- LayoutVariable property 3–93
- leaveCombo (combo.p) 7–17
- leaveLookup (lookup.p) 7–27
- leaveSelect 7–8
- LineStyle property 3–93
- LinkedFieldDataTypes property 7–42
- LinkedFieldFormats property 7–42
- linkHandles (smart.p) 2–17
- linkProperty (smart.p) 2–18
- linkRuleBuffer (toolbar.p) 6–21
- linkState (data.p) 5–51
- linkState (panel.p) 6–8
- linkState (toolbar.p) 6–21
- linkState (viewer.p) 3–70
- linkStateHandler (data.p) 5–51
- linkStateHandler (datavis.p) 3–31
- linkStateHandler (smart.p) 2–19
- linkStateHandler (viewer.p) 3–70
- ListItemPairs property 7–42
- LoadedByRouter property 8–38
- loadImage (treeview.p) 3–75
- loadPanel (panel.p) 6–8
- loadProducerSchema (b2b.p) 8–20
- loadSchema (b2b.p) 8–21
- lockRow (wbdata.p) 9–41
- LogFile property 8–38
- LogicalObjectName property 2–39
- LogicalVersion property 2–39
- LogicBuffer property 5–86
- lookup
 - prepareField 7–29
- lookup.p
 - anyKey 7–22
 - createLabel 7–23
 - destroyLookup 7–23
 - destroyObject 7–23
 - disable_UI 7–24
 - disableButton
 - disableButton (lookup.p) 7–23
 - disableField 7–24
 - displayField 7–24
 - displayLookup 7–24
 - enableButton 7–25
 - enableField 7–25
 - endMove 7–25
 - enterLookup 7–25
 - hideObject 7–26
 - initializeBrowse 7–26
 - initializeLookup 7–26
 - insertExpression 7–27
 - leaveLookup 7–27
 - newQueryString 7–28
 - newWhereClause 7–29
 - resizeObject 7–30
 - returnParentFieldValues 7–30
 - rowSelected 7–30
 - valueChanged 7–31
 - viewObject 7–31
 - whereClauseBuffer 7–32

LookupBuffer property 7–42

lookupfield.p

 notifyChildFields 7–11

 retrieveData 7–12

 returnParentFieldValues 7–12

LookupFilterValue property 7–43

LookupHandle property 7–43

LookupImage property 7–43

LookupQuery property 7–43

M

MaintenanceObject property 7–44

MaintenanceSDO property 7–44

ManualAddQueryWhere property 5–86

ManualAssignQuerySelection property
5–87

ManualSetQuerySort property 5–87

MapNameProducer property 8–38

mapNode (b2b.p) 8–21

MapObjectProducer property 8–38

mappedEntry (smart.p) 2–20

MapTypeProducer property 8–39

MarginPixels property 6–33

MasterDataObject property 4–60

MaxWidth property 3–93

Menu property 6–33

MenuMergeOrder property 6–33

messageHandler (consumer.p) 8–5

messageNumber (smart.p) 2–21

MessageType property 8–39

messaging.p

 destroyObject 8–2

 errorHandler 8–2

 initializeObject 8–2

MinHeight property 3–94, 6–34

MinWidth property 3–94, 6–34

modifyDisabledActions (toolbar.p) 6–22

ModifyFields property 3–94

modifyListProperty (smart.p) 2–22

modifyUserLinks (smart.p) 2–23

msghandler.p

 sendMessage 8–8

MultiInstanceActivated property 4–60

MultiInstanceSupported property 4–60

N

NameList property 8–39

NamespaceHandle property 8–39

NavigationSource property 4–60, 5–87

NavigationSourceEvents property 4–60,
5–87

NavigationTarget property 4–60

NavigationTargetEvents property 6–34
NavigationTargetName property 6–34
newDataObjectRow (sbo.p) 4–39
NewNode property 8–39
newQueryString (combo.p) 7–18
newQueryString (lookup.p) 7–28
newQueryString (query.p) 5–19
newQueryValidate (query.p) 5–20
newQueryWhere (query.p) 5–21
NewRow property 5–88
newRowObject (data.p) 5–52
newWhereClause (combo.p) 7–19
newWhereClause (lookup.p) 7–29
newWhereClause (query.p) 5–21
NextNodeKey property 3–95
NodeExpanded property 3–95
nodeHandle (xml.p) 8–12
nodeType (xml.p) 8–12
NotFoundError (b2b.p) 8–22
notifyChildFields (lookupfield.p) 7–11
notifyPage (containr.p) 4–16
NumDown property 3–95
numParameters (b2b.p) 8–22
NumRows property 7–44

O

ObjectEnabled property 3–95
ObjectHidden property 2–39
ObjectInitialized property 2–39
ObjectLayout property 3–96
ObjectMapping property 4–61
ObjectName property 2–39
ObjectPage property 2–40
ObjectParent property 2–40
ObjectType property 2–40
ObjectVersion property 2–40
obtainCallInfo function A–39
obtainContextForServer (containr.p) 4–17
obtainContextForServer (data.p) 5–52
obtainInMsgTarget (router.p) 8–30
obtainParamPropValue function A–43
obtainProcHandle function A–40
offEnd (browser.p) 3–55
offHome (browser.p) 3–55
okObject (containr.p) 4–18
okObject (datavis.p) 3–31
okToContinueProcedure (datavis.p) 3–32
OLEDrag property 3–96
OLEDrop property 3–96

- onChoose (panel.p) 6–8
- onChoose (toolbar.p) 6–22
- onEnd (browser.p) 3–55
- oneObjectLinks (smart.p) 2–24
- onHome (browser.p) 3–56
- onMenuDrop (toolbar.p) 6–23
- onValueChanged (toolbar.p) 6–23
- openDataQuery (data.p) 5–52
- OpenOnInit property 5–88
- openQuery (data.p) 5–53
- openQuery (query.p) 5–22
- openQuery (sbo.p) 4–39
- openQuery (webrep.p) 9–42
- OpenQuery property 5–88
- Operator property 3–96
- OperatorLongValues property 3–96
- OperatorStyle property 3–96
- OperatorViewAs property 3–97
- Optional property 7–44
- OptionalBlank property 7–44
- OptionalString property 7–45
- OutMessageSource property 8–40
- OutMessageTarget property 4–61
- outputFields (html-map.p) 9–42
- ownerElement (xml.p) 8–12

P

- pageBackward (wbtable.p) 9–42
- PageNTarget property 4–61
- pageNTargets (containr.p) 4–18
- PageSource property 4–61
- panel.p
 - activeTarget 6–4
 - countButtons 6–5
 - disableActions 6–5
 - enableActions 6–6
 - enableObject 6–6
 - hasActiveGaTarget 6–6
 - initializeObject 6–7
 - linkState 6–8
 - loadPanel 6–8
 - onChoose 6–8
 - queryPosition 6–9
 - resetCommit 6–9
 - resetNavigation 6–9
 - resetTableio 6–9
 - resetTargetActions 6–10
 - resizeObject 6–10
 - sensitizeActions 6–11
 - targetActions 6–11
 - updateState 6–11
 - viewHideActions 6–12
- PanelFrame property 6–34
- PanelLabel property 6–35
- PanelState property 6–35
- PanelType property 6–35
- ParentDataKey property 2–40
- ParentField property 7–45
- ParentFilterQuery property 7–45
- parentNode (xml.p) 8–12
- passThrough (containr.p) 4–19
- PassThroughLinks property 2–41

- Persistency property 8–40
- PhysicalObjectName property 2–41
- PhysicalTable (query.p) 5–89
- PhysicalVersion property 2–41
- PingInterval property 8–40
- populateTree (treeview.p) 3–76
- PopupOnAmbiguous property 7–45
- PopupOnNotAvail property 7–46
- PopupOnUniqueAmbiguous property 7–46
- Position character datatype
 - table type 2 A–16
- Position character table type A–16
- Position native datatype
 - Table type A–14
 - table type 1 A–14
- postCreateObjects (sbo.p) 4–39
- postCreateObjects (tvcontnr.p) 4–52
- postTransactionValidate (data.p) 5–53
- prepareErrorsForReturn (data.p) 5–53
- prepareErrorsForReturn (sbo.p) 4–40
- prepareField (combo.p) 7–19
- prepareField (lookup) 7–29
- prepareQueriesForFetch (sbo.p) 4–40
- prepareQuery (query.p) 5–22
- preTransactionValidate (data.p) 5–54
- PrimarySdoTarget property 4–61
- printToCrystal (data.p) 5–54
- Priority property 8–40
- processCDATASection (xml.p) 8–13
- processComment (xml.p) 8–13
- processDestinations (consumer.p) 8–5
- processDocument (xml.p) 8–13
- processElement (xml.p) 8–14
- processFileRefs (router.p) 8–31
- processMappings (b2b.p) 8–22
- processMessages (b2b.p) 8–23
- processRoot (xml.p) 8–14
- processText (xml.p) 8–14
- processWebRequest (wbdata.p) 9–43
- processWebRequest (wbtable.p) 9–44
- processWebRequest (webrep.p) 9–44
- produceAttributes (b2b.p) 8–23
- produceChildren (b2b.p) 8–24
- produceDocument (b2b.p) 8–24
- producer.p
 - destroyObject 8–6
 - initializeObject 8–7
 - replyHandler 8–7
 - sendMessage 8–7
- PromptLogin property 8–40
- Property property 3–97
- PropertyDialog property 2–42
- PropertyList property 5–89
- propertyType (smart.p) 2–24
- pushTableAndValidate (data.p) 5–54

Q

Query objects

column properties 5–96

query.i

deleteRecordStatic 5–13
initProp 5–17

query.p

addNotFoundMessage 5–2
addQueryWhere 5–2
assignDBRow 5–3
assignQuerySelection 5–4
bufferCopyDBToRO 5–6
closeQuery 5–7
ColumnPhysicalColumn 5–6
ColumnPhysicalTable 5–7
colValues 5–8
confirmCommit 5–8
confirmContinue 5–8
confirmUndo 5–9
dataAvailable 5–10
dbColumnDataName 5–11
dbColumnHandle 5–11
defineDataObject 5–12
fetchCurrentBatch 5–15
fetchFirst 5–13
fetchFirstBatch 5–14
fetchLast 5–13
fetchLastBatch 5–15
fetchNext 5–14
fetchNextBatch 5–14
fetchPrev 5–14
fetchPrevBatch 5–14
filterContainerHandler 5–15
firstBufferName 5–16
firstRowId 5–16
indexInformation 5–17
initializeObject 5–17
insertExpression 5–18
newQueryString 5–19
newQueryValidate 5–20
newQueryWhere 5–21
newWhereClause 5–21

openQuery 5–22
PhysicalTable 5–89
prepareQuery 5–22
removeForeignKey 5–23
removeQuerySelection 5–23
resolveBuffer 5–24
rowidWhere 5–24
rowidWhereCols 5–25
startFilter 5–25
transferDBRow 5–26
whereClauseBuffer 5–26

QueryColumns property 5–89

QueryContainer property 5–89

queryext.p

DataColumnsByTables 5–76

QueryHandle property 5–89

QueryObject property 2–42

QueryOpen property 5–90

queryOpened (select.p) 7–9

queryPosition (datavis.p) 3–32

queryPosition (panel.p) 6–9

queryPosition (sbo.p) 4–41

queryPosition (toolbar.p) 6–23

QueryPosition property 4–62, 5–90

QueryRowIden 5–90

QueryRowObject property 3–98

QuerySort property 5–90

QueryString property 5–91

QueryTables property 7–46

QueryWhere property 5–91

R

- readOffsets (html-map.p) 9–45
- RebuildOnRepos property 5–92
- receiveHandler (b2b.p) 8–24
- receiveHandler (xml.p) 8–15
- receiveReplyHandler (xml.p) 8–15
- RecordState property 3–98
- Refresh property 3–99
- refreshBrowse (browser.p) 3–56
- refreshBrowse (sbo.p) 4–41
- refreshChildDependancies (combo.p) 7–20
- refreshCombo (combo.p) 7–20
- RefreshList property 7–46
- refreshObject (select.p) 7–9
- refreshRow (data.p) 5–55
- registerCacheItem (cache.p) 4–6, 4–20
- registerLinkedObjects (sbo.p) 4–41
- registerObject (sbo.p) 4–42
- remoteCommit (data.p) 5–55
- remoteCommitTransaction (sbo.i) 4–42
- remoteFetchContainedData (sbo.p) 4–42
- remoteSendRows (data.p) 5–56
- remoteSendRows (sbo.p) 4–43
- removeAllLinks (smart.p) 2–25
- removeEntry (webrep.p) 9–45
- removeForeignKey (query.p) 5–23
- removeLink (smart.p) 2–25
- removePageNTarget (containr.p) 4–20
- removeQuerySelection (query.p) 5–23
- removeQuerySelection (sbo.p) 4–45
- removeSpace (filter.p) 3–43
- reOpenQuery (webrep.p) 9–46
- replyHandler (producer.p) 8–7
- ReplyReqList property 8–41
- ReplyRequired property 8–41
- ReplySelector property 8–41
- ReplySelectorList property 8–41
- repositionDataSource (select.p) 7–9
- RepositionDataSource property 7–47
- repositionObject (smart.p) 2–26
- repositionObject (toolbar.p) 6–24
- repositionObject (treeview.p) 3–77
- repositionRowObject (data.p) 5–57
- repositionRowObject (sbo.p) 4–45
- resetCommit (panel.p) 6–9
- resetFields (filter.p) 3–44
- resetNavigation (panel.p) 6–9
- resetQuery (sbo.p) 4–46
- resetRecord (browser.p) 3–56
- resetRecord (datavis.p) 3–33
- resetTableio (panel.p) 6–9
- resetTargetActions (panels.p) 6–10
- resetTargetActions (toolbar.p) 6–24
- resetToolbar (toolbar.p) 6–24

- resetWidgetValue (visual.p) 3–14
- resizeBrowse (browser.p) 3–57
- resizeObject (browser.p) 3–57
- resizeObject (combo.p) 7–20
- resizeObject (field.p) 7–2
- resizeObject (lookup.p) 7–30
- resizeObject (panel.p) 6–10
- resizeObject (select.p) 7–10
- resizeObject (toolbar.p) 6–25
- resizeObject (treeview.p) 3–77
- ResizeVertical property 3–99
- resizeWindow (containr.p) 4–21
- resolveBuffer (query.p) 5–24
- resortQuery (data.p) 5–58
- restartServerObject (sbo.p) 4–46
- retrieveBandsAndActions (toolbar.p) 6–25
- retrieveData (lookupfield.p) 7–12
- returnFocus (smart.p) 2–27
- returnParentFieldValues (combo.p) 7–21
- returnParentFieldValues (lookup.p) 7–30
- returnParentFieldValues (lookupfield.p) 7–12
- reviewMessages (smart.p) 2–27
- RootNodeParentKey property 3–99
- routeBytesMessage (router.p) 8–31
- routeDocument (router.p) 8–31
- routeMessage (router.p) 8–32
- router.p
 - createDocument 8–29
 - initializeObject 8–30
 - internalSchemaFile 8–30
 - obtainInMsgTarget 8–30
 - processFileRefs 8–31
 - routeBytesMessage 8–31
 - routeDocument 8–31
 - routeMessage 8–32
 - startB2BObject 8–32
- RouterSource property 8–41
- RouterTarget property 4–62
- Row property 3–99
- rowAvailable (data.p) 5–58
- rowBatchAvailable 5–59
- rowChanged (browser.p) 3–58
- rowDisplay (browser.p) 3–58
- RowIdent property 3–100, 5–92
- rowidExpression (webrep.p) 9–46
- rowidWhere (query.p) 5–24
- rowidWhereCols (query.p) 5–25
- rowNotFoundError (b2b.p) 8–25
- RowObject property 3–100
- rowObjectState (toolbar.p) 6–26
- RowObjectState property 4–62, 5–92
- RowObjectTable property 5–92
- rowObjectValidate (datalogic.i) 5–59
- RowObjUpd property 5–92
- RowObjUpdTable property 5–93
- rowSelected (lookup.p) 7–30
- rowSelected (select.p) 7–10

RowsToBatch property 5–93, 7–47
rowValues (data.p) 5–60
rowVisible (browser.p) 3–58
RunAttribute property 2–42
RunDataLogicProxy property 4–62
RunDOOptions property 4–64
runInfo (toolbar.p) 6–26
runServerObject (appserver.p) 2–32
runServerProcedure (appserver.p) 2–32

S

saveContextAndDestroy (data.p) 5–61
SavedScreenValue property 7–47
SaveSource property 3–100
sbo.i
 initDataObjectOrdering 4–36
 remoteCommitTransaction 4–42
 serverCommitTransaction 4–47
sbo.p
 addDataTarget 4–25
 addNavigationSource 4–26
 addQueryWhere 4–26
 addRow 4–27
 appendContainedObjects 4–27
 assignCurrentMappedObject 4–28
 assignMaxGuess 4–28
 assignQuerySelection 4–28
 cancelNew 4–29
 cancelRow 4–29
 canNavigate 4–30
 colValues 4–30
 commitTransaction 4–31
 confirmContinue 4–31
 containedProperties 4–8
 copyRow 4–32
 currentMappedObject 4–32
 dataAvailable 4–32
 dataObjectHandle 4–33
 deleteComplete 4–33
 deleteRow 4–33
 destroyServerObject 4–33
 endClientDataRequest 4–34
 fetchBatch 4–34
 fetchContainedData 4–34
 fetchContainedRows 4–35
 fetchDOProperties 4–35
 fetchFirst 4–35
 fetchLast 4–36
 fetchNext 4–36
 fetchPrev 4–36
 findRowWhere 4–37
 initializeObject 4–38
 initializeServerObject 4–38
 isUpdatePending 4–38
 newDataObjectRow 4–39
 openQuery 4–39
 postCreateObjects 4–39
 prepareErrorsForReturn 4–40
 prepareQueriesForFetch 4–40
 queryPosition 4–41
 refreshBrowse 4–41
 registerLinkedObjects 4–41
 registerObject 4–42
 remoteFetchContainedData 4–42
 remoteSendRow 4–43
 removeQuerySelection 4–45
 repositionRowObject 4–45
 resetQuery 4–46
 restartServerObject 4–46
 serverContainedSendRows 4–48
 serverFetchContainedData 4–49
 serverFetchContainedRows 4–49
 serverFetchDOProperties 4–50
 setPropertyList 4–50
 startServerObject 4–50
 submitRow 4–51
 undoTransaction 4–51
 updateState 4–51

schemaField (b2b.p) 8–25	refreshObject 7–9
SchemaHandle property 8–42	repositionDataSource 7–9
SchemaList property 8–42	resizeObject 7–10
SchemaLocation property 5–93	rowSelected 7–10
SchemaManager property 8–42	valueChanged 7–10
Scroll property 3–100	viewObject 7–11
ScrollRemote property 3–101	SelectedNode property 3–101
SDFFileName property 7–47	selectFirstNode (treeview.p) 3–78
SDFTemplate property 7–48	selectNode (treeview.p) 3–78
SdoForeignFields property 4–65	Selectors property 8–42
SearchField property 3–101	selectPage (containr.p) 4–21
searchTrigger (browser.p) 3–59	sendHandler (b2b.p) 8–26
Secured property 7–48	sendHandler (xml.p) 8–15
SecuredTokens property 6–35	sendMessage (b2b.p) 8–26
select.p	sendMessage (msghandler.p) 8–8
anyKey 7–3	sendMessage (producer.p) 8–7
browseHandler 7–3	sendReplyHandler (xml.p) 8–16
buildList 7–3	sendRows (data.p) 5–62
createLabel 7–4	sensitizeActions (panel.p) 6–11
dataAvailable 7–4	serverCommit (data.p) 5–63
destroyObject 7–5	serverCommitTransaction (sbo.i) 4–47
destroySelection 7–5	serverContainedSendRows (sbo.p) 4–48
disable_UI 7–6	serverFetchContainedData (sbo.p) 4–49
disableButton 7–5	serverFetchContainedRows (sbo.p) 4–49
disableField 7–5	serverFetchDOProperties (sbo.p) 4–50
enableButton 7–6	ServerFileName property 2–43
enableField 7–6	ServerOperatingMode property 2–43
endMove 7–6	serverSendRows (data.p) 5–64
enterSelect 7–7	
formattedValue 7–7	
hideObject 7–7	
initializeBrowse 7–8	
initializeObject 7–8	
initializeSelection 7–8	
queryOpened 7–9	

- ServerSubmitValidation property 5–93
- setAppService (webrep.p) 9–47
- set-attribute-list (admweb.p) 9–47
- setBuffers (webrep.p) 9–48
- setColumns (webrep.p) 9–48
- setContextFields (wbrep.p) 9–48
- setContextFields (webrep.p) 9–48
- setCurrentRowids (webrep.p) 9–49
- setDeleteTables (wbdata.p) 9–49
- setExternalJoinList (webrep.p) 9–50
- setExternalTableList (webrep.p) 9–50
- setExternalTables (wbrep.p) 9–51
- setExternalTables (webrep.p) 9–51
- setExternalWhereList (webrep.p) 9–52
- setForeignFieldList (webrep.p) 9–52
- setFrameHandle (wbdata.p) 9–53
- setLinkColumns (wbtable.p) 9–53
- setLinkColumns (webrep.p) 9–53
- setPropertyList (sbo.p) 4–50
- setQueryWhere (webrep.p) 9–54
- setSearchColumns (webrep.p) 9–54
- setServerConnection (webrep.p) 9–54
- setTableModifier (wbtable.p) 9–55
- setTableRows (wbtable.p) 9–55
- setUpdateMode (webrep.p) 9–56
- setupTTTFromSig function A–40
- setupTTTFromString function A–41
- setupTTTFromTable function A–43
- setUseColumnLabels (wbtables.p) 9–56
- setWebState (admweb.p) 9–57
- setWebToHdlr (admweb.p) 9–57
- ShareData property 5–94
- ShowBorder property 6–35
- ShowCheckBoxes property 3–101
- showDataMessages (datavis.p) 3–33
- showDataMessages (filter.p) 3–44
- showDataMessages (webrep.p) 9–58
- showDataMessagesProcedure (datavis.p) 3–34
- showMessage (smart.p) 2–28
- showMessageProcedure (smart.p) 2–28
- showQuery (combo.p) 7–21
- ShowRootLines property 3–101
- showTVCErrors (tvcontroller.p) 4–52
- showTVError (treeview.p) 3–78
- ShutDownDest property 8–42
- Signature (smart.p) 2–29
- SingleSel property 3–102
- smart.i
 - start-super-proc 2–29

smart.p
 addLink 2-2
 addMessage 2-4
 adjustTabOrder 2-5
 anyMessage 2-6
 applyEntry 2-6
 assignLinkProperty 2-7
 changeCursor 2-8
 createControls 2-8
 destroyObject 2-9
 displayLinks 2-10
 editInstanceProperties 2-10
 exitObject 2-11
 fetchMessages 2-12
 fixQueryString 2-13
 hideObject 2-16
 initializeObject 2-13
 instanceOf 2-14
 instancePropertyList 2-15
 linkHandles 2-17
 linkProperty 2-18
 linkStateHandler 2-19
 mappedEntry 2-20
 messageNumber 2-21
 modifyListProperty 2-22
 modifyUserLinks 2-23
 oneObjectLinks 2-24
 propertyType 2-24
 removeAllLinks 2-25
 removeLink 2-25
 repositionObject 2-26
 returnFocus 2-27
 reviewMessages 2-27
 showMessage 2-28
 showMessageProcedure 2-28
 Signature 2-29
 viewObject 2-29

Sort property 3-102

startB2BObject (router.p) 8-32

StartBrowseKeys property 7-48

startDataObject (webrep.p) 9-58

startDataRow (b2b.p) 8-26

startElement (b2b.p) 8-27

startFilter (query.p) 5-25

startSearch (browser.p) 3-59

startServerObject (data.p) 5-65

startServerObject (sbo.p) 4-50

start-super-proc (smart.i) 2-29

startWaitFor (consumer.p) 8-6

StaticPrefix 6-35

stopHandler (consumer.p) 8-6

storeNodeValue (b2b.p) 8-27

storeParameterNode (b2b.p) 8-28

storeParameterValue (b2b.p) 8-28

storePendingSensitivity (toolbar.p) 6-27

stripCalcs (browser.p) 3-59

submitCommit (data.p) 5-65

submitForeignKey (data.p) 5-66

submitRow (data.p) 5-67

submitRow (sbo.p) 4-51

Subscriptions property 8-42

SupportedLinks property 2-43

SupportedMessageTypes property 8-43

synchronizeProperties (data.p) 5-68

T

Table Type

position character datatype table A-16

position native datatype A-14

Table type1

position native datatype table A-14

- Table type2
 - position character datatype table A-16
- TableIOSource property 3-102
- TableIOSourceEvents property 3-102
- TableioTarget property 6-36
- TableioTargetEvents property 6-36
- TableioType property 6-36
- tableOut (data.p) 5-68
- Tables property 5-94
- targetActions (panel.p) 6-11
- targetActions (toolbar.p) 6-27
- TargetNameSpace property 8-43
- targetPage (containr.p) 4-22
- targetProcedure (b2b.p) 8-29
- Temp-table types A-14
- TimeToLive property 8-43
- timingOut (admweb.p) 9-58
- ToggleDataTargets property 3-102, 5-94
- toggleWidget (visual.p) 3-15
- toolbar (browser.p) 3-60
- toolbar (containr.p) 4-22
- toolbar (viewer.p) 3-70
- Toolbar objects
 - action properties 6-38
- Toolbar property 6-36
- toolbar.p
 - actionCanRun 6-12
 - actionCaption 6-13
 - actionCategoryIsHidden 6-13
 - actionChecked 6-14
 - actionLabel 6-14
 - actionPublishCreate 6-14
 - actionTarget 6-15
 - actionTooltip 6-15
 - buildAllMenus 6-16
 - canFindAction 6-2
 - canFindCategory 6-2
 - categoryActions 6-16
 - categoryLink 6-2
 - checkRule 6-3
 - constructMenu 6-16
 - constructToolbar 6-16
 - createMenuTable 6-17
 - createMenuTable2 6-18
 - createToolbar 6-18
 - defineAction 6-3
 - displayActions 6-4
 - filterState 6-19
 - hideObject 6-19
 - imageName 6-19
 - initAction 6-4
 - initializeObject 6-4, 6-20
 - insertMenu 6-20
 - linkRuleBuffer 6-21
 - linkState 6-21
 - modifyDisabledActions 6-22
 - onChoose 6-22
 - onMenuDrop 6-23
 - onValueChanged 6-23
 - queryPosition 6-23
 - repositionObject 6-24
 - resetTargetActions 6-24
 - resetToolbar 6-24
 - resizeObject 6-25
 - retrieveBandsAndActions 6-25
 - rowObjectState 6-26
 - runInfo 6-26
 - storePendingSensitivity 6-27
 - targetActions 6-27
 - updateActive 6-27
 - updateCategoryLists 6-28
 - updateState 6-28
 - updateStates 6-29
 - viewHideActions 6-29
 - viewObject 6-29
- ToolbarAutoSize property 6-36
- ToolbarBands property 6-37

ToolbarDrawDirection property 6–37

ToolbarHeightPx1 property 6–37

ToolbarInitialState property 6–37

ToolbarSource property 3–103

ToolbarSourceEvents property 3–103, 6–37

ToolbarTargetEvents property 6–37

ToolbarWidthPx1 property 6–38

ToolMarginPx1 property 6–38

ToolTip property 7–48

Tooltip property 3–103

TopOnly property 4–65

transferDBRow (query.p) 5–26

transferToExcel (data.p) 5–69

TranslatableProperties property 2–43

TreeDataTable property 3–103

TreeStyle property 3–104

treeview

- repositionObject 3–77

treeview.p

- addNode 3–72
- deleteNode 3–73
- deleteTree 3–74
- disableObject 3–74
- emptyTree 3–74
- enableObject 3–74
- isNodeExpanded 3–75
- loadImage 3–75
- populateTree 3–76
- resizeObject 3–77
- selectFirstNode 3–78
- selectNode 3–78
- showTVError 3–78
- updateTree 3–79

tvcontnr.p

- postCreateObjects 4–52

tvcontroller.p

- initializeObject 4–52
- showTVCErrors 4–52
- updateState 4–53

TVControllerSource property 3–104

TVControllerTarget property 3–104

TVControllerTargetEvents property 3–104

TypeName property 8–43

U

UIBMode property 2–43

unbindServer (appserver.p) 2–32

unBlankFillin (filter.p) 3–44

unBlankLogical (filter.p) 3–45

undoClientUpdate (data.p) 5–70

undoTransaction (data.p) 5–70

undoTransaction (sbo.p) 4–51

UpdatableColumns property 5–95

UpdatableColumnsByTable property 5–96

updateActive (containr.p) 4–23

updateActive (toolbar.p) 6–27

UpdateActive property 4–65

updateAddQueryWhere (data.p) 5–70

updateCategoryLists (toolbar.p) 6–28

updateData (data.p) 5–71

UpdateFromSource property 5–95

updateMode (datavis.p) 3–34

updateQueryPosition (data.p) 5–72
updateRecord (browser.p) 3–60
updateRecord (datavis.p) 3–35
updateRow (data.p) 5–72
UpdateSource property 4–65
updateState (browser.p) 3–61
updateState (data.p) 5–73
updateState (datavis.p) 3–35
updateState (panel.p) 6–11
updateState (sbo.p) 4–51
updateState (toolbar.p) 6–28
updateState (tvcontroller.p) 4–53
updateState (viewer.p) 3–71
updateStates (toolbar.p) 6–29
UpdateTarget property 3–105
UpdateTargetNames property 3–105
updateTitle (browser.p) 3–61
updateTree (treeview.p) 3–79
urlJoinParams (webrep.p) 9–59
urlLink (webrep.p) 9–60
UseBegins property 3–105
UseContains property 3–105
UseDBQualifier property 5–95
UseDTD property 8–43
UsePairedList property 7–48
UseRepository property 2–44
UserProperty property 2–44

V

validateColumns (wbdata.p) 9–60
validateColumnValue (webrep.p) 9–61
validateFields (datavis.p) 3–35
ValidateOnLoad property 8–44
ValidKey property 3–105
ValueChanged (browser.p) 3–61
valueChanged (combo.p) 7–21
valueChanged (lookup.p) 7–31
valueChanged (select.p) 7–10
valueChanged (viewer.p) 3–71
ViewAs property 7–49
viewer.p
 addRecord 3–62
 cancelRecord 3–63
 copyRecord 3–64
 disableFields 3–65
 displayFields 3–66
 enableFields 3–68
 fieldModified 3–68
 initializeObject 3–69
 linkState 3–70
 linkStateHandler 3–70
 toolbar 3–70
 updateState 3–71
 valueChanged 3–71
 viewRecord 3–71
ViewerLinkedFields property 7–49
ViewerLinkedWidgets property 7–49
viewHideActions (panel.p) 6–12
viewHideActions (toolbar.p) 6–29
viewObject (browser.p) 3–62
viewObject (combo.p) 7–22

- viewObject (containr.p) 4–24
- viewObject (lookup.p) 7–31
- viewObject (select.p) 7–11
- viewObject (smart.p) 2–29
- viewObject (toolbar.p) 6–29
- viewPage (containr.p) 4–24
- viewRecord (viewer.p) 3–71
- viewWidget (visual.p) 3–16
- visual.p
 - applyLayout 3–2
 - assignFocusedWidget 3–2
 - assignWidgetValue 3–3
 - assignWidgetValueList 3–4
 - blankWidget 3–5
 - disableObject 3–6
 - disableRadioButton 3–6
 - disableWidget 3–7
 - enableObject 3–8
 - enableRadioButton 3–8
 - enableWidget 3–9
 - formattedWidgetValue 3–10
 - formattedWidgetValueList 3–10
 - hideWidget 3–12
 - highlightWidget 3–12
 - initializeObject 3–14
 - resetWidgetValue 3–14
 - toggleWidget 3–15
 - viewWidget 3–16
 - widgetHandle 3–17
 - widgetIsBlank 3–17
 - widgetIsFocused 3–18
 - widgetIsModified 3–18
 - widgetIsTrue
 - widgetIsTrue (visual.p) 3–19
 - widgetValue 3–20
 - widgetValueList 3–21
- VisualBlank property 3–106

W

- WaitForObject property 4–65
- Waiting property 8–44
- wbdata.p
 - assignFields 9–10
 - columnValMsg 9–18
 - deleteBuffer 9–19
 - deleteRow 9–20
 - exclusiveLockBuffer 9–24
 - getDeleteTables 9–30
 - getFrameHandle 9–31
 - lockRow 9–41
 - processWebRequest 9–43
 - setDeleteTables 9–49
 - setFrameHandle 9–53
 - validateColumns 9–60
- wbrep.p
 - HTMLSetFocus 9–38
 - setContextFields 9–48
 - setExternalTables 9–51
- wbtable.p
 - addColumnLink 9–2
 - addTDModifier 9–6
 - assignTDModifier 9–11
 - columnTDModifier 9–17
 - fetchLast 9–26
 - fetchNext 9–26
 - fetchPrev 9–27
 - getTableRows 9–34
 - HTMLColumn 9–38
 - HTMLTable 9–39
 - pageBackward 9–42
 - processWebRequest 9–44
 - setLinkColumns 9–53
 - setTableModifier 9–55
 - setTableRows 9–55
 - setUseColumnLabels 9–56

- webrep.p
 - addContextFields 9–5
 - addSearchCriteria 9–5
 - anyMessage 9–6
 - assignColumnForma 9–7
 - assignColumnHelp 9–8
 - assignColumnLabel 9–8
 - assignColumnWidt 9–9
 - assignExtentAttribute 9–9
 - bufferHandle 9–13
 - columnDataType 9–13
 - columnFormat 9–13
 - columnHandle 9–14
 - columnHelp 9–14
 - columnHTMLName 9–15
 - columnLabel 9–16
 - columnReadOnly 9–16
 - columnStringValue 9–16
 - destroyDataObject 9–21
 - destroyObject 9–21
 - disconnectObject 9–21
 - extentAttribute 9–24
 - extentValue 9–25
 - fetchCurrent 9–25
 - fetchFirst 9–25
 - fetchLast 9–26
 - fetchNext 9–27
 - fetchPrev 9–27
 - fieldExpression 9–28
 - getContextFields 9–29
 - getCurrentRowids 9–30
 - getForeignFieldList 9–31
 - getNavigationMode 9–31
 - getQueryEmpty 9–33
 - getQueryWhere 9–33
 - getRowids 9–33
 - getSearchColumns 9–33
 - getServerConnection 9–34
 - getTableRowids 9–34
 - getTables 9–35
 - getUpdateMode 9–35
 - HTMLAlert 9–37
 - joinExternalTables 9–40
 - joinForeignFields 9–41
 - openQuery 9–42
 - processWebRequest 9–44
 - removeEntry 9–45
 - reOpenQuery 9–46
 - rowidExpression 9–46
 - setAppService 9–47
 - setBuffers 9–48
 - setColumns 9–48
 - setContextFields 9–48
 - setCurrentRowids 9–49
 - setExternalJoinList 9–50
 - setExternalTableList 9–50
 - setExternalTables 9–51
 - setExternalWhereList 9–52
 - setForeignFieldList 9–52
 - setLinkColumns 9–53
 - setQueryWhere 9–54
 - setSearchColumns 9–54
 - setServerConnection 9–54
 - setUpdateMode 9–56
 - showDataMessages 9–58
 - startDataObject 9–58
 - urlJoinParams 9–59
 - urlLink 9–60
 - validateColumnValue 9–61
- whereClauseBuffer (combo.p) 7–22
- whereClauseBuffer (lookup.p) 7–32
- whereClauseBuffer (query.p) 5–26
- widgetBlank (visual.p) 3–17
- widgetFocused (visual.p) 3–18
- widgetHandle (visual.p) 3–17
- widgetIsModified (visual.p) 3–18
- widgetValue (visual.p) 3–20
- widgetValueList (visual.p) 3–21
- Width property 3–106
- WindowFrameHandle property 4–66
- WindowTitleField property 3–106
- WordIndexedFields property 5–96

X

xml.p	
assignNodeValue	8–9
createAttribute	8–9
createElement	8–10
createNode	8–10
createText	8–11
deleteDocument	8–11
destroyObject	8–11
nodeHandle	8–12
nodeType	8–12
	ownerElement 8–12
	parentNode 8–12
	processCDATASection 8–13
	processComment 8–13
	processDocument 8–13
	processElement 8–14
	processRoot 8–14
	processText 8–14
	receiveHandler 8–15
	receiveReplyHandler 8–15
	sendHandler 8–15
	sendReplyHandler 8–16