



Progress Dynamics. Managers 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: 4481
Item Number: 103616;V2.1B

Contents

Preface	xiii
Purpose	xiii
Audience	xiii
Organization of this manual	xiii
How to use this manual	xiv
Public and private APIs	xv
Typographical conventions	xv
Syntax notation	xvi
Progress messages	xx
1. Configuration File Manager	1-1
1.1 Overview	1-2
1.2 Temp-tables used by Configuration File Manager	1-3
1.3 Configuration File Manager APIs	1-5
1.3.1 detectFileType	1-5
1.3.2 expandTokens	1-5
1.3.3 findFile	1-6
1.3.4 getCodePath	1-6
1.3.5 getExpandablePropertyValue	1-6
1.3.6 getManagerHandle	1-7
1.3.7 getPhysicalSessionType	1-7
1.3.8 getProcedureHandle	1-7
1.3.9 getSessionParam	1-8
1.3.10 initializeSession	1-8
1.3.11 isConfigManRunning	1-8
1.3.12 isICFRunning	1-9
1.3.13 obtainCFMTables	1-9
1.3.14 obtainRegistryKeys	1-9
1.3.15 parseConfig	1-10

1.3.16	propertyExpander	1-10
1.3.17	sessionShutdown	1-10
1.3.18	setICFIsRunning	1-11
1.3.19	setSessionParam	1-11
1.3.20	subscribeAll	1-11
2.	Connection and Service Type Managers	2-1
2.1	Overview	2-2
2.2	Temp-tables used by Connection and Service Type Managers	2-4
2.3	Connection Manager APIs	2-5
2.3.1	connectService	2-5
2.3.2	disconnectService	2-5
2.3.3	getConnectionParams	2-5
2.3.4	getConnectionString	2-6
2.3.5	getPhysicalService	2-6
2.3.6	getServiceHandle	2-6
2.3.7	getServiceList	2-7
2.3.8	getServiceSTManager	2-7
2.3.9	getServiceType	2-7
2.3.10	getServiceTypeManager	2-8
2.3.11	initializeServices	2-8
2.3.12	isConnected	2-8
2.3.13	obtainConnectionTables	2-9
2.3.14	plipShutdown	2-9
2.3.15	reconnectService	2-9
2.3.16	registerService	2-10
2.3.17	setServiceTypeManager	2-10
2.4	AppServer Connection Manager APIs	2-11
2.4.1	appServerConnect	2-11
2.4.2	appServerDisconnect	2-11
2.4.3	connectService	2-11
2.4.4	definedPartitions	2-12
2.4.5	disconnectService	2-12
2.4.6	findServiceRecord	2-12
2.4.7	getASConnectString	2-13
2.4.8	getConnectionParams	2-13
2.4.9	getConnectionString	2-13
2.4.10	getJMSPartitions	2-14
2.4.11	getJMSPtnInfo	2-14
2.4.12	getPartitionsByType	2-14
2.4.13	getPhysicalService	2-15
2.4.14	getServiceField	2-15
2.4.15	getServiceHandle	2-15
2.4.16	getServiceList	2-16

2.4.17	initializeSelf	2-16
2.4.18	isConnected	2-16
2.4.19	isDefaultService	2-17
2.4.20	loadPartitionInfo	2-17
2.4.21	parseConnectionParams	2-17
2.4.22	plipShutdown	2-17
2.4.23	reconnectService	2-18
2.4.24	registerService	2-18
2.4.25	savePartitionInfo	2-18
2.4.26	security_prompt	2-19
2.4.27	setServiceHandle	2-19
2.5	Database Connection Manager APIs	2-20
2.5.1	connectService	2-20
2.5.2	disconnectService	2-20
2.5.3	findServiceRecord	2-20
2.5.4	getConnectionParams	2-21
2.5.5	getConnectionString	2-21
2.5.6	getPhysicalService	2-21
2.5.7	getServiceField	2-22
2.5.8	getServiceHandle	2-22
2.5.9	getServiceList	2-22
2.5.10	isConnected	2-23
2.5.11	isDefaultService	2-23
2.5.12	parseConnectionParams	2-23
2.5.13	plipShutdown	2-24
2.5.14	registerService	2-24
2.5.15	setServiceHandle	2-24
3.	Customization Manager	3-1
3.1	Overview	3-2
3.2	Customization Manager APIs	3-3
3.2.1	getClientResultCodes	3-3
3.2.2	getCustomisationTypesPrioritised	3-3
3.2.3	getReferenceLanguage	3-3
3.2.4	getReferenceLoginCompany	3-4
3.2.5	getReferenceSystem	3-4
3.2.6	getReferenceUIType	3-4
3.2.7	getReferenceUser	3-4
3.2.8	getReferenceUserCategory	3-5
3.2.9	getSessionCustomisationReferences	3-5
3.2.10	getSessionResultCodes	3-5
3.2.11	InitializeObject	3-5
3.2.12	PlipShutdown	3-6
3.2.13	setClientResultCodes	3-6

3.2.14	setSessionResultCodes	3-7
4.	General Manager	4-1
4.1	Overview	4-2
4.2	Temp-tables used by General Manager	4-3
4.3	General Manager APIs	4-7
4.3.1	cacheEntity	4-7
4.3.2	cacheEntityMapping	4-7
4.3.3	checkIfOverlaps	4-8
4.3.4	convertTimeToInteger	4-9
4.3.5	createFolder	4-9
4.3.6	formatPersonDetails	4-10
4.3.7	getDBsForImportedEntities	4-10
4.3.8	getDBVersion	4-11
4.3.9	getDumpName	4-11
4.3.10	getEntityCacheBuffer	4-12
4.3.11	getEntityDescription	4-12
4.3.12	getEntityDetail	4-13
4.3.13	getEntityDisplayField	4-13
4.3.14	getEntityExists	4-14
4.3.15	getEntityFieldCacheBuffer	4-14
4.3.16	getEntityTableName	4-15
4.3.17	getHighKey	4-15
4.3.18	getInternalEntries	4-16
4.3.19	getKeyField	4-16
4.3.20	getLanguageText	4-16
4.3.21	getNextSequenceValue	4-18
4.3.22	getObjField	4-18
4.3.23	getOEMCode	4-19
4.3.24	getOEMDescription	4-19
4.3.25	getPropertyFromList	4-19
4.3.26	getRecordCheckAudit	4-20
4.3.27	getRecordCheckComment	4-20
4.3.28	getRecordDetail	4-21
4.3.29	getRecordUserProp	4-21
4.3.30	getSequenceConfirmation	4-22
4.3.31	getSequenceExist	4-22
4.3.32	getSequenceMask	4-22
4.3.33	getSiteNumber	4-23
4.3.34	getStatusObj	4-23
4.3.35	getStatusShortDesc	4-24
4.3.36	getStatusTLA	4-24
4.3.37	getTableDumpName	4-24
4.3.38	getTableInfo	4-25

4.3.39	getTableInfoObj	4-25
4.3.40	getUpdatableTableInfo	4-26
4.3.41	getUpdatableTableInfoObj	4-26
4.3.42	getUserSourceLanguage	4-27
4.3.43	haveOutstandingUpdates	4-27
4.3.44	listLookup	4-27
4.3.45	plipShutdown	4-28
4.3.46	refreshMnemonicsCache	4-28
4.3.47	setPropertyValueInList	4-28
4.3.48	setWidgetAttribute	4-29
4.3.49	updateTableViaSDO	4-29
4.3.50	validateEntityMnemonic	4-30
5.	Localization Manager	5-1
5.1	Overview	5-2
5.2	Temp-tables used by Localization Manager	5-3
5.3	Localization Manager APIs	5-4
5.3.1	buildClientCache	5-4
5.3.2	buildWidgetTable	5-4
5.3.3	clearClientCache	5-5
5.3.4	getTranslation	5-6
5.3.5	multiTranslation	5-7
5.3.6	plipShutdown	5-8
5.3.7	receiveCacheClient	5-8
5.3.8	translatePhrase	5-8
5.3.9	translateWidgetTable	5-9
5.3.10	updateTranslations	5-9
6.	Profile Manager	6-1
6.1	Overview	6-2
6.2	Temp-tables used by Profile Manager	6-3
6.3	Profile Manager APIs	6-4
6.3.1	buildClientCache	6-4
6.3.2	checkProfileDataExists	6-4
6.3.3	clearClientCache	6-5
6.3.4	deleteSessionProfile	6-6
6.3.5	getProfileData	6-6
6.3.6	getProfileTTHandle	6-7
6.3.7	plipShutdown	6-7
6.3.8	receiveProfileCache	6-8
6.3.9	setProfileData	6-8
6.3.10	updateCacheToDb	6-10

7.	Referential Integrity Manager	7-1
7.1	Overview	7-2
7.2	Temp-tables used by Referential Integrity Manager	7-3
7.3	Referential Integrity Manager APIs	7-5
7.3.1	versionData	7-5
8.	Repository Managers	8-1
8.1	Overview	8-2
8.2	Temp-tables used by Repository Managers	8-4
8.2.1	Temp-tables used by Repository Manager	8-5
8.2.2	Temp-tables used by Repository Design Manager	8-8
8.3	Repository Manager APIs	8-12
8.3.1	calculateObjectPaths	8-12
8.3.2	classHasAttribute	8-14
8.3.3	classIsA	8-14
8.3.4	clearClientCache	8-14
8.3.5	createClassCache	8-15
8.3.6	destroyClassCache	8-15
8.3.7	extractRootFile	8-16
8.3.8	getCacheClassBuffer	8-16
8.3.9	getCacheLinkBuffer	8-17
8.3.10	getCacheObjectBuffer	8-17
8.3.11	getCachePageBuffer	8-17
8.3.12	getClassChildren	8-18
8.3.13	getClassFromInstance	8-18
8.3.14	getClientCacheDir	8-18
8.3.15	getCurrentLogicalName	8-19
8.3.16	getMappedFilename	8-19
8.3.17	getObjectNames	8-20
8.3.18	getObjectSuperProcedure	8-20
8.3.19	getToolBarBandActions	8-21
8.3.20	IsA	8-22
8.3.21	plipShutdown	8-22
8.3.22	resolveResultCodes	8-22
8.3.23	startDataObject	8-23
8.3.24	storeAttributeValues	8-23
8.4	Repository Design Manager APIs	8-24
8.4.1	changeObjectInstance	8-24
8.4.2	changeObjectType	8-25
8.4.3	classHasAttribute	8-26
8.4.4	clearDesignCache	8-26
8.4.5	copyObjectMaster	8-27
8.4.6	generateCalculatedField	8-28
8.4.7	generateClassCache	8-28

8.4.8	generateDataFields	8-29
8.4.9	generateDataLogicObject	8-30
8.4.10	generateDataObject	8-31
8.4.11	generateDynamicBrowse	8-32
8.4.12	generateDynamicSDF	8-33
8.4.13	generateDynamicViewer	8-34
8.4.14	generateEntityInstances	8-35
8.4.15	generateEntityObject	8-36
8.4.16	generateSBODataLogicObject	8-37
8.4.17	generateSDOInstances	8-38
8.4.18	generateVisualObject	8-39
8.4.19	getBufferDbName	8-40
8.4.20	getProductModuleList	8-40
8.4.21	getSchemaQueryHandle	8-41
8.4.22	getWidgetSizeFromFormat	8-42
8.4.23	insertObjectInstance	8-42
8.4.24	insertObjectLinks	8-43
8.4.25	insertObjectMaster	8-44
8.4.26	insertObjectPage	8-46
8.4.27	insertUiEvents	8-47
8.4.28	ObjectExists	8-47
8.4.29	plipShutdown	8-47
8.4.30	prepareObjectName	8-48
8.4.31	removeAttributeValues	8-49
8.4.32	removeObject	8-50
8.4.33	removeObjectInstance	8-51
8.4.34	removeObjectPage	8-52
8.4.35	removePageInstance	8-53
8.4.36	removeUiEvents	8-54
8.4.37	setQualifiedTableName	8-54
9.	Security Manager	9-1
9.1	Overview	9-2
9.2	Temp-Tables used by Security Manager	9-3
9.3	Security Manager APIs	9-6
9.3.1	areFieldsCached	9-6
9.3.2	areTokensCached	9-6
9.3.3	authenticateUser	9-7
9.3.4	cacheGlobalSecurityAllocations	9-7
9.3.5	cacheGlobalSecurityStructures	9-7
9.3.6	changePassword	9-8
9.3.7	checkUser	9-9
9.3.8	clearClientCache	9-10
9.3.9	createGroupAllocation	9-11

9.3.10	createGroupFromUser	9-11
9.3.11	fieldandtokenSecurityCheck	9-12
9.3.12	fieldSecurityCheck	9-13
9.3.13	fieldSecurityGet	9-13
9.3.14	getContainerIcons	9-14
9.3.15	getFieldSecurity	9-14
9.3.16	getMandatoryTables	9-15
9.3.17	getSecurityControl	9-16
9.3.18	menuItemSecurityCheck	9-16
9.3.19	menuStructureSecurityCheck	9-17
9.3.20	objectSecurityCheck	9-17
9.3.21	plipShutdown	9-18
9.3.22	rangeSecurityCheck	9-18
9.3.23	receiveCacheSessionSecurity	9-19
9.3.24	tableSecurityCheck	9-19
9.3.25	tokenSecurityCheck	9-20
9.3.26	tokenSecurityGet	9-20
9.3.27	updateUserAllocations	9-21
9.3.28	userLoginOrganisations	9-22
9.3.29	userSecurityCheck	9-22
10.	Session Manager	10-1
10.1	Overview	10-2
10.2	Temp-tables used by Session Manager	10-4
10.3	Session Manager APIs	10-7
10.3.1	activateSession	10-7
10.3.2	askQuestion	10-7
10.3.3	contextHelp	10-9
10.3.4	createLinks	10-9
10.3.5	deletePersistentProc	10-10
10.3.6	fixQueryString	10-10
10.3.7	getActionUnderway	10-10
10.3.8	getCurrentLogicalName	10-11
10.3.9	getGlobalControl	10-11
10.3.10	getHelp	10-12
10.3.11	getInternalEntryExists	10-12
10.3.12	getLoginUserInfo	10-12
10.3.13	getPersistentProcs	10-13
10.3.14	getPropertyList	10-13
10.3.15	helpAbout	10-14
10.3.16	helpContents	10-14
10.3.17	helpHelp	10-14
10.3.18	helpTopics	10-15
10.3.19	htmlHelpKeywords	10-15

10.3.20	htmlHelpTopic	10–16
10.3.21	increaseFrameforPopup	10–16
10.3.22	isObjQuoted	10–16
10.3.23	killPlips	10–17
10.3.24	killProcedure	10–18
10.3.25	launchContainer	10–18
10.3.26	LaunchExternalProcess	10–20
10.3.27	launchProcedure	10–20
10.3.28	notifyUser	10–22
10.3.29	plipShutdown	10–22
10.3.30	resizeLookupFrame	10–23
10.3.31	resizeNormalFrame	10–23
10.3.32	resizeSDFFrame	10–24
10.3.33	runLookup	10–24
10.3.34	sendEmail	10–25
10.3.35	setActionUnderway	10–26
10.3.36	setAttributesInObject	10–27
10.3.37	setPropertyList	10–27
10.3.38	setReturnValue	10–28
10.3.39	setSecurityForDynObjects	10–28
10.3.40	showMessages	10–29
10.3.41	showWarningMessages	10–30
10.3.42	translateWidgets	10–31
10.3.43	updateErrorLog	10–31
10.3.44	updateHelp	10–32
10.3.45	widgetWalk	10–32
11.	User Interface Manager	11–1
11.1	Overview	11–2
11.2	User Interface Manager APIs	11–4
11.2.1	escapeData	11–4
11.2.2	setClientAction	11–4
12.	Web Request Manager	12–1
12.1	Overview	12–2
12.2	Web Request Manager APIs	12–3
12.2.1	processRequest	12–3
Index		Index–1

Tables

Table 1–1:	Configuration File Manager files	1–2
Table 1–2:	Temp-tables defined in af\sup2\afxmlcftgtt.i	1–3
Table 1–3:	Temp-table defined in af\sup2\afstnode.i	1–4
Table 2–1:	Connection Manager files	2–2
Table 2–2:	AppServer Connection Manager files	2–3
Table 2–3:	Database Connection Manager files	2–3
Table 2–4:	Temp-tables defined in af\app\afconttdef.i	2–4
Table 2–5:	Temp-tables defined in af\sup2\afsrvtype.i	2–4
Table 3–1:	Customization Manager files	3–2
Table 4–1:	General Manager files	4–2
Table 4–2:	Temp-table defined in af\app\afgenmgrp.i	4–3
Table 4–3:	Temp-table defined in af\app\gsmstttcch.i	4–4
Table 4–4:	Temp-table defined in af\app\gsttenmn.i	4–5
Table 5–1:	Localization Manager files	5–2
Table 5–2:	Temp-tables defined in af\app\afsttranslation.i	5–3
Table 5–3:	Temp-tables defined in adm2\stttranslate.i	5–3
Table 6–1:	Profile Manager files	6–2
Table 6–2:	Temp-table defined in af\app\afstprofiledata.i	6–3
Table 7–1:	Referential Integrity Manager files	7–2
Table 7–2:	Temp-tables defined in ry\app\ryrisrvr.p	7–3
Table 8–1:	Repository Manager files	8–3
Table 8–2:	Repository Design Manager files	8–4
Table 8–3:	Temp-tables defined in ry\inc\getobject.i	8–5
Table 8–4:	Temp-tables defined in ry\inc\ryrepatset.i	8–7
Table 8–5:	Temp-tables defined in ry\app\rymenufunc.i	8–7
Table 8–6:	Other temp-tables used by the Repository Manager	8–8
Table 8–7:	Temp-tables defined in ry\inc\ryreplnset.i	8–8
Table 8–8:	Temp-tables defined in ry\app\rygenomngp.p	8–9
Table 8–9:	Temp-tables defined in ry\app\rydesmgrp.i	8–11
Table 9–1:	Security Manager files	9–2
Table 9–2:	Temp-tables defined in af\app\afsecmgrp.i	9–3
Table 9–3:	Temp-tables defined in af\app\afsecttdef.i	9–4
Table 9–4:	Temp-tables defined in af\app\afstsecurityctrl.i	9–5
Table 10–1:	Session Manager files	10–3
Table 10–2:	Temp-tables defined in af\app\afsesmgrp.i	10–4
Table 10–3:	Temp-table defined in af\app\logintt.i	10–5
Table 10–4:	Temp-tables defined in af\app\afsttpersist.i	10–5
Table 10–5:	Temp-table defined in af\app\afsttglobalctrl.i	10–6
Table 10–6:	Other temp-tables used by the Session Manager	10–6
Table 11–1:	User Interface Manager files	11–3
Table 12–1:	Web Request Manager files	12–2

Preface

Purpose

This manual provides a basic reference to the APIs that a developer uses to make calls to the Progress Dynamics® environment managers.

Audience

This reference is intended for developers who design applications with Progress Dynamics.

Organization of this manual

The reference is organized as follows:

[Chapter 1, “Configuration File Manager”](#)

Describes the Configuration File Manager and its APIs.

[Chapter 2, “Connection and Service Type Managers”](#)

Describes the Connection and Service Type Managers and their APIs.

[Chapter 3, “Customization Manager”](#)

Describes the Customization Manager and its APIs.

[Chapter 4, “General Manager”](#)

Describes the General Manager and its APIs.

[Chapter 5, “Localization Manager”](#)

Describes the Localization Manager and its APIs.

[Chapter 6, “Profile Manager”](#)

Describes the Profile Manager and its APIs.

[Chapter 7, “Referential Integrity Manager”](#)

Describes the Referential Integrity Manager and its APIs.

[Chapter 8, “Repository Managers”](#)

Describes the Repository and Repository Design Managers and their APIs.

[Chapter 9, “Security Manager”](#)

Describes the Security Manager and its APIs.

[Chapter 10, “Session Manager”](#)

Describes the Session Manager and its APIs.

[Chapter 11, “User Interface Manager”](#)

Describes the User Interface Manager and its APIs.

[Chapter 12, “Web Request Manager”](#)

Describes the Web Request Manager and its APIs.

How to use this manual

This reference discusses the application program interface (API) for the Progress Dynamics™ environment managers. Each chapter discusses a manager or group of related managers. Each chapter provides a general description of the manager, the temp-tables it uses, and the APIs that your applications should use to interact with the manager.

Because most of the files referenced are under the ICF directory, most of the paths are relative to `DLC\src\icf`. The paths for files that are not under the ICF directory are given relative to `DLC\src`.

Public and private APIs

The Progress Dynamics Managers contain many internal procedures and functions, but only a subset of them should be directly accessed by your applications. To differentiate between APIs that should be accessed and those that should not be accessed, the header comment for each API will include something like one of the following lines:

```
ACCESS_LEVEL=PUBLIC
```

or

```
ACCESS_LEVEL=PRIVATE
```

APIs marked PUBLIC are subject to a formal deprecation policy. APIs marked PRIVATE are intended solely to support the framework's operation. They might be radically restructured or removed at any time to improve the framework's performance. There will be no formal notification of changes to PRIVATE APIs. APIs that have not been marked as either PUBLIC or PRIVATE are being evaluated. They will be marked in future releases.

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 <i>aggregate expression</i>
--

- 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**).

Configuration File Manager

The Configuration File Manager handles the initial startup of the Progress Dynamics® environment.

This chapter covers the following subjects:

- [Overview](#)
- [Temp-tables used by Configuration File Manager](#)
- [Configuration File Manager APIs](#)

1.1 Overview

The Configuration File Manager is a server-side procedure. It is the first procedure started in the Progress Dynamics environment and initializes the startup environment. It scans the configuration file, `icfconfig.xml`, for the definition of the configuration that is starting. Using the information from that definition, it starts the Connection Manager, any specified Service Type Managers, and the Session Manager.

The Configuration File Manager is not a configurable manager. Because it is responsible for starting the Progress Dynamics environment, you should never alter its behavior.

You can retrieve the handle of the Configuration File Manager by running the following code:

```
ASSIGN hConfigFileManager =  
DYNAMIC-FUNCTION("getManagerHandle":U, INPUT "ConfigFileManager":U).
```

Table 1–1 shows the files that contain the Configuration File Manager’s code.

Table 1–1: Configuration File Manager files

Type	Path
Client-side wrapper	NA
Server-side wrapper	NA
Main code	af\app\afxm1cfgp.p
Included files	adecomm_adetool.i adm2\globals.i af\sup2\aficfcheck.i af\sup2\afbtnode.i af\sup2\afxm1cfgtt.i af\sup2\afxm1replctrl.i ry\app\ryplipmain.i ry\app\ryplipkill.i ry\app\ryplipsetu.i ry\app\ryplipshut.i

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in the *Progress Dynamics Programming Handbook*.

1.2 Temp-tables used by Configuration File Manager

The Configuration File Manager uses several temp-tables to store information used in starting the framework.

Table 1–2 describes the temp-tables defined in `afxm1cfgtt.i`.

Table 1–2: Temp-tables defined in `afxm1cfgtt.i`

(1 of 2)

Temp-table	Fields (data types)
ttManager	cSessionType (Character) iOrder (Integer) cManagerName (Character) cFileName (Character) cHandleName (Character) cSuperOf (Character) hHandle (Handle) iUniqueID (Integer) iDBOrder (Integer) iCFGOrder (Integer) lUpdated (Logical) lDelete (Logical)
ttParam	cOption (Character) cValue (Character) cDispValue (Character)
ttProperty	cSessionType (Character) cProperty (Character) cValue (Character) lUpdated (Logical) lDelete (Logical)

Table 1–2: Temp-tables defined in af\sup2\afxmfcfgtt.i

(2 of 2)

Temp-table	Fields (data types)
ttService	cSessionType (Character) iOrder (Integer) cServiceType (Character) cServiceName (Character) cPhysicalService (Character) cConnectParams (Character) lDefaultService (Logical) lCanRunLocal (Logical) iStartOrder (Integer) iDBOrder (Integer) iCFGOrder (Integer) lUpdated (Logical) lDelete (Logical)
ttSession{&ttSessionExt}	cSessionType (character) {&session-table-fields}

NOTE: The preprocessor inserted in the temp-table name allows you to define separate temp-tables for different sessions by passing in different values to definition statements.

The Configuration File Manager uses the temp-table described in [Table 1–3](#) to store information about the nodes in the configuration file, icfconfig.xml:

Table 1–3: Temp-table defined in af\sup2\afbtnode.i

Temp-table	Fields (data types)
ttNode	cNode (Character) cValue (Character) iNodeLevel (Integer) lDelete (Logical)

1.3 Configuration File Manager APIs

This section lists the APIs that you can use with the Configuration File Manager.

1.3.1 detectFileType

This function determines the type of file from the filename. The valid file types are:

- U = URL
- N = UNC Filename (\\Machine\share\directory\file)
- D = DOS\Windows (D:\directory\file)
- X = UNIX Filename

Location: af\app\afxm1cfgp.p

Parameters:

INPUT pcFileName AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the validateConfigFile procedure in af\app\afxm1cfgp.p.

1.3.2 expandTokens

This function replaces tokens in the string with the values from the session parameters. Tokens in the string are in the form: #<token>#. The replacement values are derived from the getSessionParam function.

Location: af\app\afxm1cfgp.p

Parameters:

INPUT pcString AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the propertyExpander procedure in af\app\afxm1cfgp.p.

1.3.3 findFile

This function takes a filename and tries to find the file on disk using the `_start_in_dir` and `_framework` directories.

Location: `af\app\afxmlcfgp.p`

Parameters:

INPUT `pcFileName` AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the `applyUpdates` procedure in `install\prc\insessupdp.p`.

1.3.4 getCodePath

This function returns a string containing the path for a passed-in file.

Location: `af\app\afxmlcfgp.p`

Parameters:

INPUT `pcFileName` AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the `startProcedure` procedure in `af\app\afxmlcfgp.p`.

1.3.5 getExpandablePropertyValue

This function obtains the value of an expandable property.

Location: `af\app\afxmlcfgp.p`

Parameters:

INPUT `pcProperty` AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the `propertyExpander` procedure in `af\app\afxmlcfgp.p`.

1.3.6 getManagerHandle

This function gets the handle of an already started manager. It returns an unknown value (?) if the manager has not been started.

Location: af\app\afxm1cfgp.p

Parameters:

INPUT pcManagerName AS CHARACTER

Returns: HANDLE

Notes: None

Examples: See the sessionShutdown procedure in af\app\afxm1cfgp.p.

1.3.7 getPhysicalSessionType

This function determines the session's physical session type. If the value has not previously been set as a parameter, it is set here.

Location: af\app\afxm1cfgp.p

Parameters: None

Returns: CHARACTER

Notes: None

Examples: See the initializeSession procedure in af\app\afxm1cfgp.p.

1.3.8 getProcedureHandle

This function determines the handle to a running procedure (if it is running) and returns the handle. It returns an unknown value (?) if the handle cannot be determined.

Location: af\app\afxm1cfgp.p

Parameters:

INPUT pcFileName AS CHARACTER

Returns: HANDLE

Notes: This function assumes that there is only one running copy of the procedure.

Examples: See the startProcedure procedure in af\app\afxm1cfgp.p.

1.3.9 getSessionParam

This function returns the value associated with an option.

Location: af\app\afxm1cfgp.p

Parameters:

INPUT pcOption AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the main block in af\app\afxm1cfgp.p.

1.3.10 initializeSession

This procedure controls setting all the global environment settings.

Location: af\app\afxm1cfgp.p

Parameters:

INPUT pcICFParam AS CHARACTER

Notes: None

Examples: See the main block in icf\icfstart.p.

1.3.11 isConfigManRunning

This function checks if the Configuration Manager is running. If it is, the function returns a value of TRUE. This allows users to find out if the Configuration File Manager is running without having to obtain its handle. Users can simply call it using a DYNAMIC-FUNCTION call.

Location: af\app\afxm1cfgp.p

Parameters: None

Returns: LOGICAL

Notes: None

Examples: See the start-super-proc procedure in af\sup2\afsrvtype.i.

1.3.12 isICFRunning

This function checks if the Progress Dynamics environment is established and running. It returns a value of TRUE if the environment is running.

If the global variable glICFIsRunning is unknown, this function checks the values of the global variables that contain the essential Progress Dynamics managers. If glICFIsRunning is not unknown, this function returns the value of that variable. This enables the user to override the standard definition of what constitutes a running Progress Dynamics session.

Location: af\app\afxm1cfgp.p

Parameters: None

Returns: LOGICAL

Notes: This function assumes that opaque handles are switched on.

Examples: See the main block in icf\icfstart.p.

1.3.13 obtainCFMTables

This procedure returns the handles to all the temp-tables that the Connection File Manager has populated after reading the configuration file.

Location: af\app\afxm1cfgp.p

Parameters:

OUTPUT phParam AS HANDLE

OUTPUT phManager AS HANDLE

Notes: This procedure deliberately does not return all the temp-tables as some of them are working tables that should not have their values displayed.

Examples: See the getHandles procedure in af\cod2\afsessinfo.w.

1.3.14 obtainRegistryKeys

This procedure parses the RegistryKeys property to find the properties that contain registry keys that need to be loaded. It then parses each of those keys and loads the key values.

Location: af\app\afxm1cfgp.p

Parameters: None

Notes: None

Examples: See the initializeSession procedure in af\app\afxm1cfgp.p.

1.3.15 parseConfig

This procedure takes the handle to `icfconfig.xml` and retrieves the contents of the requested session types into the configuration temp-tables.

Location: `af\app\afxm1cfgp.p`

Parameters:

INPUT `phXDoc` AS HANDLE

Handle to the X Document that has the info.

INPUT `pcSessType` AS CHARACTER

Session Type to read in. A blank value retrieves all the data.

Notes: None

Examples: See the `initializeSession` procedure in `af\app\afxm1cfgp.p`.

1.3.16 propertyExpander

This procedure expands all the properties in the expand list using the appropriate method.

Location: `af\app\afxm1cfgp.p`

Parameters: None

Notes: None

Examples: See the `initializeSession` procedure in `af\app\afxm1cfgp.p`.

1.3.17 sessionShutdown

This procedure cleans up the session and ensures that all the managers are properly closed down.

Location: `af\app\afxm1cfgp.p`

Parameters: None

Notes: None

Examples: See the main block in `af\app\afxm1cfgp.p`.

1.3.18 setICFIsRunning

This function exposes the gLICFIsRunning variable to other procedures. This enables the variable to be set and retrieved to determine if the Progress Dynamics environment has been properly established.

Location: af\app\afxm1cfgp.p

Parameters:

INPUT plRunning AS LOGICAL

Returns: LOGICAL

Notes: None

1.3.19 setSessionParam

This function sets the value of a parameter in the ttParam table.

Location: af\app\afxm1cfgp.p

Parameters:

INPUT pcOption AS CHARACTER

INPUT pcValue AS CHARACTER

Returns: LOGICAL

Notes: None

Examples: See the initializeSession procedure in af\app\afxm1cfgp.p.

1.3.20 subscribeAll

This procedure subscribes a procedure to the events that need to be trapped from the Configuration File Manager startup.

Location: af\app\afxm1cfgp.p

Parameters:

INPUT phSourceProc AS HANDLE

INPUT phTargetProc AS HANDLE

Notes: None

Examples: See the main block in icf\icfstart.p.

Connection and Service Type Managers

The Connection and Service Type Managers handle the connections to databases, Progress AppServers™, and other services for the Progress Dynamics environment.

This chapter covers the following subjects:

- [Overview](#)
- [Temp-tables used by Connection and Service Type Managers](#)
- [Connection Manager APIs](#)
- [AppServer Connection Manager APIs](#)
- [Database Connection Manager APIs](#)

2.1 Overview

The Connection and Service Type Managers work together to establish and maintain connections to services in the Progress Dynamics environment, such as AppServers and databases. The Connection Manager uses the Service Type Manager for a service type as a device driver to handle the details of connections to all services of that type. All these managers run on the server-side of the environment.

At session start, the Configuration File Manager retrieves the session type data from the configuration file. The Connection Manager is always the next manager started. After it has started, any necessary Service Type Managers start. The precedence for starting the Service Type Managers is generally AppServer, Database, and then any other.

You can retrieve the handle of these Managers by running code like the following example for the Connection Manager:

```
ASSIGN hConnectionManager =  
DYNAMIC-FUNCTION("getManagerHandle":U, INPUT "ConnectionManager":U).
```

Table 2–1 shows the files that contain the Connection Manager’s code.

Table 2–1: Connection Manager files

Type	Path
Client-side wrapper	NA
Server-side wrapper	NA
Main code	af\app\afconmgrp.p
Included files	af\sup2\afglobals.i af\app\afconttdef.i

The Progress Dynamics framework comes with prebuilt Service Type Managers for AppServers and databases. In the configuration file, these managers are referred to as the AppServer Connection Manager and Database Connection Manager.

You can also create custom Service Type Managers to handle other services. You can find a template for new Service Type Managers in af\app\afsvrconmgrp.p.

Table 2–2 shows the files that contain the AppServer Connection Manager’s code.

Table 2–2: AppServer Connection Manager files

Type	Path
Client-side wrapper	NA
Server-side wrapper	NA
Main code	af\app\afasconmgrp.p
Included files	adecomm_adetool.i adecomm\appsrvtt.i af\sup2\aficfcheck.i af\sup2\afsrvtype.i
Other important files	af\app\afservicetype.p

Table 2–3 shows the files that contain the Database Connection Manager’s code.

Table 2–3: Database Connection Manager files

Type	Path
Client-side wrapper	NA
Server-side wrapper	NA
Main code	af\app\afdbconmgrp.p
Included files	af\sup2\aficfcheck.i af\sup2\afsrvtype.i
Other important files	af\app\afservicetype.p

For more information on these Managers, see the chapter on using the Progress Dynamics Managers in the *Progress Dynamics Programming Handbook*.

2.2 Temp-tables used by Connection and Service Type Managers

The managers use the temp-tables to store information about services and service types. The temp-tables use the general form described in [Table 2-4](#).

Table 2-4: Temp-tables defined in `af\app\afconttdef.i`

Temp-table	Fields (data types)
<code>tt{&ttPrefix}Service</code>	<code>cServiceName</code> (Character) <code>cPhysicalService</code> (Character) <code>cServiceType</code> (Character) <code>cConnectParams</code> (Character) <code>IDefaultService</code> (Logical)
<code>tt{&ttPrefix}ServiceType</code>	<code>cServiceType</code> (Character) <code>cSTProcName</code> (Character) <code>hSTManager</code> (Handle) <code>IUseHandle</code> (Logical)

NOTE: The preprocessor inserted in the temp-table name allows you to define separate Service and Service Type temp-tables for different managers by passing in different values to the definition statements in `af\app\afconttdef.i`.

The managers also use the temp-table described in [Table 2-5](#).

Table 2-5: Temp-tables defined in `af\sup2\afsrvtype.i`

Temp-table	Fields (data types)
<code>ttService</code>	<code>cServiceName</code> (Character) <code>cPhysicalService</code> (Character) <code>cConnectParams</code> (Character) <code>cServiceHandle</code> (Character) <code>IDefaultService</code> (Logical) {&ServiceTypeFields}

NOTE: The preprocessor allows you to add fields that are specific to each service type to the table.

2.3 Connection Manager APIs

This section lists the APIs that you can use with the Connection Manager.

2.3.1 connectService

This procedure establishes the connection to the required physical service.

Location: af\app\afconmgr.p

Parameters:

INPUT pcServiceName AS CHARACTER

OUTPUT pcHandle AS CHARACTER

Notes: None

Examples: See the initializeServices procedure in af\app\afconmgr.p.

2.3.2 disconnectService

This procedure disconnects a physical service.

Location: af\app\afconmgr.p

Parameters:

INPUT pcServiceName AS CHARACTER

Notes: None

Examples: See the processDB procedure in install\prc\iniutil.p.

2.3.3 getConnectionParams

This function returns the connection parameters for a logical service.

Location: af\app\afconmgr.p

Parameters:

INPUT pcServiceName AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the getConnectionString function in af\app\afservicetype.p.

2.3.4 **getConnectionString**

This function retrieves the connection string for a logical service.

Location: af\app\afconmgrp.p

Parameters:

INPUT pcServiceName AS CHARACTER

Returns: CHARACTER

Notes: None

2.3.5 **getPhysicalService**

This function returns the physical service associated with a logical service name.

Location: af\app\afconmgrp.p

Parameters:

INPUT pcServiceName AS CHARACTER

Returns: CHARACTER

Notes: None

2.3.6 **getServiceHandle**

This function gets the handle to a specific service.

Location: af\app\afconmgrp.p

Parameters:

INPUT pcServiceName AS CHARACTER

Returns: CHARACTER

Notes: None

2.3.7 getServiceList

This function dynamically runs the getServiceList function for the procedure that should make the current connection.

Location: af\app\afconmgr.p

Parameters:

INPUT pcServiceType AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the initializeObject procedure in af\obj2\afgendatov.w.

2.3.8 getServiceSTManager

This function returns the Service Type Manager handle for a specified service name.

Location: af\app\afconmgr.p

Parameters:

INPUT pcServiceName AS CHARACTER

Returns: HANDLE

Notes: None

Examples: See the isConnected function in af\app\afconmgr.p.

2.3.9 getServiceType

This function returns the service type of a known service.

Location: af\app\afconmgr.p

Parameters:

INPUT pcServiceName AS CHARACTER

Returns: CHARACTER

Notes: None

2.3.10 getServiceTypeManager

This function determines the handle to the procedure that manages a particular service.

Location: af\app\afconmgrp.p

Parameters:

INPUT pcServiceType AS CHARACTER

Returns: HANDLE

Notes: None

Examples: See the isConnected function in af\app\afconmgrp.p.

2.3.11 initializeServices

This procedure reads the contents of the table passed using the temp-table. It then buffer-copies the contents to the ttService table.

Location: af\app\afconmgrp.p

Parameters:

INPUT phService AS HANDLE

INPUT p1Connect AS LOGICAL

Notes: None

Examples: See the initializeSession procedure in af\app\afxm1cfgp.p.

2.3.12 isConnected

This function determines whether or not a specific service is connected.

Location: af\app\afconmgrp.p

Parameters:

INPUT pcServiceName AS CHARACTER

Returns: LOGICAL

Notes: None

Examples: See the registerService procedure in af\app\afconmgrp.p.

2.3.13 obtainConnectionTables

This procedure retrieves the handles for the temp-tables that hold the data from the Connection Manager and Service Manager.

Location: af\app\afconmgr.p

Parameters:

OUTPUT phServiceType AS HANDLE

The handle of ttServiceType.

OUTPUT phService AS HANDLE

The handle of ttService.

Notes: None

Examples: See the main block in af\app\afapping.p.

2.3.14 plipShutdown

This procedure runs on shutdown of the manager to ensure everything closes properly.

Location: af\app\afconmgr.p

Parameters: None

Notes: None

2.3.15 reconnectService

This procedure attempts to reconnect a service after it fails.

Location: af\app\afconmgr.p

Parameters:

INPUT pcServiceName AS CHARACTER

OUTPUT pcHandle AS CHARACTER

Notes: None

2.3.16 registerService

This procedure registers a service with the Connection Manager and the appropriate Service Type Manager.

Location: af\app\afconmgr.p

Parameters:

INPUT phService AS HANDLE

Notes: None

Examples: See the initializeServices procedure in af\app\afconmgr.p.

2.3.17 setServiceTypeManager

This function creates a record in the service type temp-table that registers the persistent procedure responsible for managing each service type.

Location: af\app\afconmgr.p

Parameters:

INPUT phServiceProc AS HANDLE

Returns: LOGICAL

Notes: None

Examples: See the initializeSession procedure in af\app\afxm1cfgp.p.

2.4 AppServer Connection Manager APIs

This section lists the APIs that you can use with the AppServer Connection Manager.

2.4.1 appServerConnect

This procedure used to be called in the as-utils. It is here for backward compatibility.

Location: af\app\afasconmgr.p

Parameters:

INPUT partition_name AS CHARACTER

INPUT security_prompt AS LOGICAL

INPUT app_server_info AS CHARACTER

OUTPUT conn_hdl AS HANDLE

Notes: None

2.4.2 appServerDisconnect

This procedure used to be called in the as-utils. It is here for backward compatibility.

Location: af\app\afasconmgr.p

Parameters:

INPUT partition_name AS CHARACTER

Notes: The AppServer should only be disconnected with disconnectService.

2.4.3 connectService

This procedure connects a physical service for a given service name. This procedure is a required entry point for the Connection Manager.

Location: af\app\afasconmgr.p

Parameters:

INPUT pcServiceName AS CHARACTER

OUTPUT pcHandle AS CHARACTER

Notes: This call uses the af\sup2\aficfcheck.i include file.

Examples: See the appServerConnect procedure in af\app\afasconmgr.p.

2.4.4 definedPartitions

This function used to be called in the as-utils. It is here for backward compatibility.

Location: af\app\afasconmgrp.p

Parameters: None

Returns: CHARACTER

Notes: None

2.4.5 disconnectService

This procedure disconnects a physical service. This procedure is a required entry point for the Connection Manager.

Location: af\app\afasconmgrp.p

Parameters:

INPUT pcServiceName AS CHARACTER

Notes: None

Examples: See the plipShutdown procedure in af\app\afasconmgrp.p.

2.4.6 findServiceRecord

This function finds the service record for a specific service name.

Location: af\app\afservicetype.p

Parameters:

INPUT pcServiceName AS CHARACTER

The service name.

INPUT pcLock AS CHARACTER

Returns: HANDLE

Notes: None

2.4.7 **getASConnectString**

This function used to be called in the as-utils. It is here for backward compatibility.

Location: af\app\afasconmgrp.p

Parameters:

BUFFER bAppSrvTT FOR AppSrv-TT

Returns: CHARACTER

Notes: None

2.4.8 **getConnectionParams**

This function returns the connection parameter field associated with a logical service name.

Location: af\app\afservicetype.p

Parameters:

INPUT pcServiceName AS CHARACTER

The logical service name.

Returns: CHARACTER

Notes: None

2.4.9 **getConnectionString**

This function returns the connection string used to connect to a logical service name.

Location: af\app\afservicetype.p

Parameters:

INPUT pcServiceName AS CHARACTER

The logical service name.

Returns: CHARACTER

Notes: None

2.4.10 **getJMSPartitions**

This function used to be called in the as-utils. It is here for backward compatibility.

Location: af\app\afasconmgrp.p

Parameters: None

Returns: CHARACTER

Notes: None

2.4.11 **getJMSPtnInfo**

This function used to be called in the as-utils. It is here for backward compatibility.

Location: af\app\afasconmgrp.p

Parameters:

INPUT cPartition AS CHARACTER

Returns: CHARACTER

Notes: None

2.4.12 **getPartitionsByType**

This function used to be called in the as-utils. It is here for backward compatibility.

Location: af\app\afasconmgrp.p

Parameters:

INPUT cType AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the definedPartitions function in af\app\afasconmgrp.p.

2.4.13 **getPhysicalService**

This function returns the name of a physical service associated with a logical service name.

Location: af\app\afservicetype.p

Parameters:

INPUT pcServiceName AS CHARACTER

The logical service name.

Returns: CHARACTER

Notes: None

2.4.14 **getServiceField**

This function obtains a handle to a field on the ttService table.

Location: af\app\afservicetype.p

Parameters:

INPUT pcServiceName AS CHARACTER

The logical service name.

INPUT pcFieldName AS CHARACTER

The name of the field.

Returns: HANDLE

Notes: None

2.4.15 **getServiceHandle**

This function returns the handle to a the specific service name.

Location: af\app\afservicetype.p

Parameters:

INPUT pcServiceName AS CHARACTER

The logical service name.

Returns: CHARACTER

Notes: None

2.4.16 **getServiceList**

This function returns a CHR(3) delimited list of logical service names from the ttServiceType temp-table.

Location: af\app\afservicetype.p

Parameters: None

Returns: CHARACTER

Notes: None

2.4.17 **initializeSelf**

This procedure gets the handle of af\app\afdynuser.p, starting it persistently if necessary.

Location: af\app\afasconmgr.p

Parameters: None

Notes: None

Examples: See the main block in af\app\afasconmgr.p.

2.4.18 **isConnected**

This function determines whether the requested service type is connected.

Location: af\app\afasconmgr.p

Parameters:

INPUT pcServiceName AS CHARACTER

Returns: LOGICAL

Notes: None

Examples: See the connectService procedure in af\app\afasconmgr.p.

2.4.19 isDefaultService

This function determines if a logical service is a default service.

Location: af\app\afservicetype.p

Parameters:

INPUT pcServiceName AS CHARACTER

The logical service name.

Returns: LOGICAL

Notes: None

2.4.20 loadPartitionInfo

This procedure used to be called in the as-utils. It is here for backward compatibility.

Location: af\app\afasconmgr.p

Parameters: None

Notes: None

2.4.21 parseConnectionParams

This function parses the pcParams variable and returns the connection string.

Location: af\app\afasconmgr.p

Parameters:

INPUT pcParams AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the registerService procedure in af\app\afasconmgr.p.

2.4.22 plipShutdown

This procedure cleans up the AppServer connections when the AppServer Connection Manager shuts down.

Location: af\app\afasconmgr.p

Parameters: None

Notes: None

2.4.23 reconnectService

This procedure attempts to reconnect a session after it fails.

Location: af\app\afasconmgrp.p

Parameters:

INPUT pcServiceName AS CHARACTER

OUTPUT pcHandle AS CHARACTER

Notes: None

2.4.24 registerService

This procedure takes a buffer containing a service and creates an entry in the local service temp-table if necessary. Otherwise, it attempts to disconnect the connected service and resets the service data. This procedure is a required entry point for the Connection Manager.

Location: af\app\afservicetype.p

Parameters:

INPUT phServiceBuff AS HANDLE

The handle of the buffer that containing the service.

Notes: None

2.4.25 savePartitionInfo

This procedure used to be called in the as-utils. It is here for backward compatibility.

Location: af\app\afasconmgrp.p

Parameters: None

Notes: None

2.4.26 security_prompt

This procedure used to be called in the as-utils. It is here for backward compatibility.

Location: af\app\afasconmgrp.p

Parameters:

OUTPUT 1Cancel AS LOGICAL

Notes: None

2.4.27 setServiceHandle

This function sets the value of particular service handle.

Location: af\app\afservicetype.p

Parameters:

INPUT pcServiceName AS CHARACTER

The logical service name.

INPUT pcServiceHandle AS CHARACTER

The service handle.

Returns: LOGICAL

Notes: None

2.5 Database Connection Manager APIs

This section lists the APIs that you can use with the Database Connection Manager.

2.5.1 connectService

This procedure connects a physical service for a given service name. This procedure is a required entry point for the Connection Manager.

Location: af\app\afdbconmgr.p

Parameters:

INPUT pcServiceName AS CHARACTER

OUTPUT pcHandle AS CHARACTER

Notes: This call uses the af\sup2\aficfcheck.i include file.

2.5.2 disconnectService

This procedure disconnects a physical service. This procedure is a required entry point for the Connection Manager.

Location: af\app\afdbconmgr.p

Parameters:

INPUT pcServiceName AS CHARACTER

Notes: None

2.5.3 findServiceRecord

This function finds the service record for a specific service name.

Location: af\app\afservicetype.p

Parameters:

INPUT pcServiceName AS CHARACTER

The service name.

INPUT pcLock AS CHARACTER

Returns: HANDLE

Notes: None

2.5.4 **getConnectionParams**

This function returns the connection parameter field associated with a logical service name.

Location: af\app\afservicetype.p

Parameters:

INPUT pcServiceName AS CHARACTER

 The logical service name.

Returns: CHARACTER

Notes: None

2.5.5 **getConnectionString**

This function overrides the super procedure to ensure that the input string starts with the Logical Database Name startup parameter (-ld).

Location: af\app\afdbconmgr.p

Parameters:

INPUT pcServiceName AS CHARACTER

 The logical service name.

Returns: CHARACTER

Notes: None

2.5.6 **getPhysicalService**

This function returns the name of a physical service associated with a logical service name.

Location: af\app\afservicetype.p

Parameters:

INPUT pcServiceName AS CHARACTER

 The logical service name.

Returns: CHARACTER

Notes: None

2.5.7 **getServiceField**

This function obtains a handle to a field on the ttService table.

Location: af\app\afservicetype.p

Parameters:

INPUT pcServiceName AS CHARACTER

 The logical service name.

INPUT pcFieldName AS CHARACTER

 The name of the field.

Returns: HANDLE

Notes: None

2.5.8 **getServiceHandle**

This function returns the handle to a the specific service name.

Location: af\app\afservicetype.p

Parameters:

INPUT pcServiceName AS CHARACTER

 The logical service name.

Returns: CHARACTER

Notes: None

2.5.9 **getServiceList**

This function returns a CHR(3) delimited list of logical service names from the ttServiceType temp-table.

Location: af\app\afservicetype.p

Parameters: None

Returns: CHARACTER

Notes: None

2.5.10 isConnected

This function determines whether the requested service type is connected.

Location: af\app\afdbconmgr.p

Parameters:

INPUT pcServiceName AS CHARACTER

Returns: LOGICAL

Notes: None

Examples: See the connectService procedure in af\app\afdbconmgr.p.

2.5.11 isDefaultService

This function determines if a logical service is a default service.

Location: af\app\afservicetype.p

Parameters:

INPUT pcServiceName AS CHARACTER

The logical service name.

Returns: LOGICAL

Notes: None

2.5.12 parseConnectionParams

This function parses pcParams and returns the connection string.

Location: af\app\afdbconmgr.p

Parameters:

INPUT pcParams AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the getConnectionString procedure in af\app\afservicetype.p.

2.5.13 **plipShutdown**

This procedure cleans up the database connections when the Database Connection Manager shuts down.

Location: af\app\afdbconmgr.p

Parameters: None

Notes: None

2.5.14 **registerService**

This procedure takes a buffer containing a service and creates an entry in the local service temp-table if necessary. Otherwise, it attempts to disconnect the connected service and resets the service data. This procedure is a required entry point for the Connection Manager.

Location: af\app\afservicetype.p

Parameters:

INPUT phServiceBuff AS HANDLE

The handle of the buffer that containing the service.

Notes: None

2.5.15 **setServiceHandle**

This function sets the value of particular service handle.

Location: af\app\afservicetype.p

Parameters:

INPUT pcServiceName AS CHARACTER

The logical service name.

INPUT pcServiceHandle AS CHARACTER

The service handle.

Returns: LOGICAL

Notes: None

Customization Manager

The Customization Manager determines what customizations should be applied to the user interface (UI) in a given situation.

This chapter covers the following subjects:

- [Overview](#)
- [Customization Manager APIs](#)

3.1 Overview

The Customization Manager coordinates the application of UI customizations to your application. The manager retrieves a priority list for the session and uses it to determine the order in which customizations should be applied. It then builds and caches a set of customization codes that apply to the current client session. When the client requests information from the Repository Manager, the customization codes are used to retrieve and properly display the correct objects.

In a distributed environment, the Customization Manager is started on both the client and server sides. There is a wrapper program for each side that either sets the global variable, `server-side`, to YES for the server-side version or to NO for the client-side version. Both programs then include the main code of the manager from the `ry\app\rycsmngrp.i` include file. The differences in how the manager operates on each side are triggered by checking the value of the `server-side` variable.

You can retrieve the handle of the Customization Manager by running the following code:

```
ASSIGN hCustomizationManager =  
DYNAMIC-FUNCTION("getManagerHandle":U, INPUT "CustomizationManager":U).
```

Table 3–1 shows the files that contain the Customization Manager’s code.

Table 3–1: Customization Manager files

Type	Path
Client-side wrapper	ry\app\rycusclntp.p
Server-side wrapper	ry\app\rycusssrvrp.p
Main code	ry\app\rycsmngrp.i
Included files	adm2\globals.i ry\app\rydefrescd.i

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in the *Progress Dynamics Programming Handbook*.

3.2 Customization Manager APIs

This section lists the APIs that you can use with the Customization Manager.

3.2.1 **getClientResultCodes**

This function returns result codes for the client session. When running on the AppServer, it retrieves the result codes for the currently connected client. When running on the client, it returns the session result codes.

Location: ry\app\rycusmgrp.i

Parameters: None

Returns: CHARACTER

Notes: None

Examples: See the main block in ry\app\rygetmensp.p.

3.2.2 **getCustomisationTypesPrioritised**

This function returns the prioritized customization types for a session.

Location: ry\app\rycusmgrp.i

Parameters: None

Returns: CHARACTER

Notes: None

Examples: See the main block in af\app\apppingp.p.

3.2.3 **getReferenceLanguage**

This function returns the current reference for the language.

Location: ry\app\rycusmgrp.i

Parameters: None

Returns: CHARACTER

Notes: None

3.2.4 **getReferenceLoginCompany**

This function returns the current reference for the login company. The reference is the login company code.

Location: ry\app\rycusrmgrp.i

Parameters: None

Returns: CHARACTER

Notes: None

3.2.5 **getReferenceSystem**

This function returns the current reference for the system.

Location: ry\app\rycusrmgrp.i

Parameters: None

Returns: CHARACTER

Notes: None

3.2.6 **getReferenceUIType**

This function returns the current reference for the session UI type.

Location: ry\app\rycusrmgrp.i

Parameters: None

Returns: CHARACTER

Notes: None

3.2.7 **getReferenceUser**

This function returns the current reference for the user.

Location: ry\app\rycusrmgrp.i

Parameters: None

Returns: CHARACTER

Notes: None

3.2.8 **getReferenceUserCategory**

This function returns the current reference for the user category.

Location: ry\app\rycusmgrp.i

Parameters: None

Returns: CHARACTER

Notes: Since the information is only available on the server, this call cannot run on the client.

3.2.9 **getSessionCustomisationReferences**

This function returns the session customization references.

Location: ry\app\rycusmgrp.i

Parameters: None

Returns: CHARACTER

Notes: None

Examples: See the main block in af\app\appping.p.

3.2.10 **getSessionResultCodes**

This function returns the result codes currently applicable for a session.

Location: ry\app\rycusmgrp.i

Parameters: None

Returns: CHARACTER

Notes: None

Examples: See the main block in af\app\appping.p.

3.2.11 **InitializeObject**

This procedure subscribes the Customization Manager to the ICFCFM_LoginComplete event.

Location: ry\app\rycusmgrp.i

Parameters: None

Notes: None

3.2.12 PlipShutdown

This procedure closes any super procedures of the Customization Manager.

Location: ry\app\rycusrmgrp.i

Parameters: None

Notes: None

3.2.13 setClientResultCodes

This procedure is used to set client result codes for the session. If running on the AppServer, only the clientSessionResultCodes parameter is set for the session. If running on the client, sessionResultCodes is set in the session parameter table and the clientSessionResultCodes session property is set.

Location: ry\app\rycusrmgrp.i

Parameters:

INPUT pcResultCodes AS CHARACTER

INPUT plSetOnAppserver AS LOGICAL

Notes: plSetOnAppserver should always be YES. The only situation where a NO would be passed is when it has been set on the AppServer already and is positively known to be correct.

Examples: See the setSessionResultCodes procedure in ry\app\rycusrmgrp.i.

3.2.14 **setSessionResultCodes**

This procedure sets the result codes for the current session. If run on the client, the AppServer needs to be synchronized as well. To take care of this, the procedure runs setClientResultCodes.

Location: ry\app\rycusmgrp.i

Parameters:

INPUT pcResultCodes AS CHARACTER

INPUT plSetOnAppserver AS LOGICAL

Notes: plSetOnAppserver only applies when running in a client session. By default, it should be set to YES to ensure the AppServer is synchronized when the client result codes change. However, if the result codes have already been set on the AppServer and are known to be in sync with what is being set, a NO can be passed to avoid an AppServer hit to synchronize them.

Examples: See the ICFCFM_LoginComplete procedure in ry\app\rycusmgrp.i.

General Manager

The General Manager manages information that is needed on both the client and the server for a Progress Dynamics application running in a distributed environment.

This chapter covers the following subjects:

- [Overview](#)
- [Temp-tables used by General Manager](#)
- [General Manager APIs](#)

4.1 Overview

The General Manager manages information that your business logic needs whether it is running on the client or server side of the Progress Dynamics environment. Procedures defined in this manager are referenced extensively by applications. Packaging these procedures in a persistent procedure significantly improves performance on both the client and the server, in comparison to referencing them individually as external procedures.

In a distributed environment, the General Manager is started on both the client and server sides. There is a wrapper program for each side that either sets the global variable, `server-side`, to YES for the server-side version or to NO for the client-side version. Both programs then include the main code of the manager from the `af\app\afgenmgrp.i` include file. The differences in how the manager operates on each side are triggered by checking the value of the `server-side` variable.

The General Manager’s handle is stored in the global shared variable, `gshGenManager`.

[Table 4–1](#) shows the files that contain the General Manager’s code.

Table 4–1: General Manager files

Type	Path
Client-side wrapper	af\app\afgenc1ntp.p
Server-side wrapper	af\app\afgensrvrp.p
Main code	af\app\afgenmgrp.i
Included files	adm2\globals.i af\app\afgencchentmapp.i af\app\afgengsgetenmnpp.i af\app\afgengsgetgscedp.i af\app\gsmstttcch.i af\app\gsttenmn.i af\sup2\afcheckerr.i af\sup2\aferrortxt.i af\sup2\afrun2.i
Other important files	af\app\gsgetstatp.p

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in the *Progress Dynamics Programming Handbook*.

4.2 Temp-tables used by General Manager

The General Manager uses several temp-tables to cache frequently needed information.

The General Manager uses the temp-tables described in [Table 4–2](#) to cache information about users and filter sets. The ttUser temp-table is a copy of the Repository gsm_user table.

Table 4–2: Temp-table defined in aflapplafgenmgrp.i

(1 of 2)

Temp-table	Fields (data types)
ttUser	user_obj (Decimal) user_category_obj (Decimal) user_full_name (Character) user_login_name (Character) user_creation_date (Date) user_creation_time (Integer) profile_user (Logical) created_from_profile_user_obj (Decimal) external_userid (Integer) user_password (Character) password_minimum_length (Integer) password_preexpired (Logical) password_fail_count (Integer) password_fail_date (Date) password_fail_time (Integer) password_creation_date (Date) password_creation_time (Integer) password_expiry_date (Date) password_expiry_time (Integer) update_password_history (Logical) check_password_history (Logical) last_login_date (Date) last_login_time (Integer) disabled (Logical) language_obj (Decimal) password_expiry_days (Integer) maintain_system_data (Logical) default_login_company_obj (Decimal) user_email_address (Character) development_user (Logical) security_group (Logical) default_security_group (Logical)

Table 4–2: Temp-table defined in af\app\afgenmgrp.i

(2 of 2)

Temp-table	Fields (data types)
ttFilterSetClause	filter_set_code (Character) entity_list (Character) buffer_list (Character) additional_arguments (Character) filter_set_clause (Character)

Table 4–3 describes the temp-table defined in af\app\gsmstttcch.i, which is used to cache status information.

Table 4–3: Temp-table defined in af\app\gsmstttcch.i

Temp-table	Fields (data types)
ttStatusCache	tStatusObj (Decimal) tCategoryType (Character) tCategoryGroup (Character) tCategorySubGroup (Character) tStatusShortDesc (Character) tStatusDescription (Character) tStatusTLA (Character)

The General Manager also uses the temp-tables described in [Table 4–4](#) to store information about entity display fields and entity mnemonics. The ttEntityDisplayField and ttEntityMnemonic temp-tables are defined, respectively, like the Repository's gsc_entity_display_field and gsc_entity_mnemonic tables with additional fields. The ttEntityMap temp-table is defined like the ttEntityMnemonic temp-table.

Table 4–4: Temp-table defined in a\app\gsttenmn.i

(1 of 2)

Temp-table	Fields (data types)
ttEntityDisplayField	entity_display_field_obj (Decimal) entity_mnemonic (Character) display_field_name (Character) display_field_order (Integer) display_field_label (Character) display_field_column_label (Character) display_field_format (Character) display_field_datatype (Character)
ttEntityMnemonic	entity_mnemonic (Character) entity_mnemonic_short_desc (Character) entity_mnemonic_description (Character) auto_properform_strings (Logical) entity_mnemonic_label_prefix (Character) entity_mnemonic_obj (Character) entity_description_field (Character) entity_description_procedure (Character) entity_narration (Character) entity_object_field (Character) table_has_object_field (Logical) entity_key_field (Character) table_prefix_length (Integer) field_name_separator (Character) auditing_enabled (Logical) version_data (Logical) deploy_data (Logical) entity_dbname (Character) replicate_entity_mnemonic (Character) replicate_key (Character) scm_field_name (Character) reuse_deleted_keys (Logical) hasAudit (logical) hasComment (logical) hasAutoComment (logical)

Table 4–4: Temp-table defined in a\app\gsttenmn.i (2 of 2)

Temp-table	Fields (data types)
ttEntityMap	entity_mnemonic (Character) entity_mnemonic_short_desc (Character) entity_mnemonic_description (Character) auto_properform_strings (Logical) entity_mnemonic_label_prefix (Character) entity_mnemonic_obj (Character) entity_description_field (Character) entity_description_procedure (Character) entity_narration (Character) entity_object_field (Character) table_has_object_field (Logical) entity_key_field (Character) table_prefix_length (Integer) field_name_separator (Character) auditing_enabled (Logical) version_data (Logical) deploy_data (Logical) entity_dbname (Character) replicate_entity_mnemonic (Character) replicate_key (Character) scm_field_name (Character) reuse_deleted_keys (Logical) hasAudit (logical) hasComment (logical) hasAutoComment (logical)

4.3 General Manager APIs

This section lists the APIs that you can use with the General Manager.

4.3.1 **cacheEntity**

This procedure caches the passed-in entity. The procedure follows this process:

1. Retrieves the entity object and instances from the Repository Manager cache.
2. Populates the ttEntityMnemonic table from the Repository Manager cache.
3. Populates the ttDataField table from the Repository Manager cache.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcEntityToCache AS CHARACTER

INPUT pcTableToCache AS CHARACTER

Notes: None

Examples: See the cacheEntityDisplayFields procedure in af\app\afgenmgrp.i.

4.3.2 **cacheEntityMapping**

This procedure caches entity mapping information.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcEntity AS CHARACTER

INPUT pcTable AS CHARACTER

OUTPUT TABLE FOR ttEntityMap

Notes: None

Examples: See the cacheEntity procedure in af\app\afgenmgrp.i.

4.3.3 checkIfOverlaps

This procedure checks if any record in a specified table exists that would overlap with a specified date range.

Location: af\app\afgenmnggrp.i

Parameters:

INPUT pcTable AS CHARACTER

Name of table to do search on.

INPUT pcKeyField AS CHARACTER

Name of the keyfield to use in search. (Most probably foreign-key.)

INPUT pcFromField AS CHARACTER

Name of the 'from field', that is, from_date \ admission_date.

INPUT pcToField AS CHARACTER

Name of the 'to field', that is, to_date \ discharge_date.

INPUT pdCurrentRecordObj AS DECIMAL

Obj value of current record - just to ensure that you do not compare values to the same record when modifying.

INPUT pdKeyValue AS DECIMAL

Value of the keyfield, generally the obj number.

INPUT ptFromValue AS DATE

Value to compare from.

INPUT ptToValue AS DATE

Value to compare to.

INPUT pcAdditionalWhere AS CHARACTER

Additional WHERE clause that can be added to the query if needed.

OUTPUT p1Overlap AS LOGICAL

Logical value specifying if overlapping was found.

OUTPUT ptOverlapFrom AS DATE

Null(?) if no overlapping, otherwise from date of overlapping record.

OUTPUT ptOverlapTo AS DATE

Null(?) if no overlapping, otherwise to date of overlapping record.

Notes: The pcAdditionalWhere parameter creates a buffer for pcTable, thus if you passed in gsm_person, the created buffer will be bgsm_person. Remember to use the prefixed 'b' in your criteria specification

Examples: See the main block in af\app\afgenchkifovrlpp.p.

4.3.4 convertTimeToInteger

This function takes a character string and converts it into an integer value.

Location: af\app\afgenmnggrp.i

Parameters:

INPUT pcTime AS CHARACTER

Returns: INTEGER

Notes: The pcTime parameter can be specified as “HH:MM”, “HH:MM:SS”, or “HHMMSS”.

Notes: See the getDataValue function in ry\tem\rytentimesdf.w.

4.3.5 createFolder

This function creates a folder, if it does not exist.

Location: af\app\afgenmnggrp.i

Parameters:

INPUT pcFolderName AS CHARACTER

Returns: LOGICAL

Notes: None

Examples: See the main block in af\app\afgendlogp.p.

4.3.6 formatPersonDetails

This function formats the details for a specified person in a useful string to represent the data similarly with viewers in the system.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcLastName AS CHARACTER

INPUT pcFirstName AS CHARACTER

INPUT pcInitials AS CHARACTER

Returns: CHARACTER

Notes: None

4.3.7 getDBsForImportedEntities

This procedure scans through the entity mnemonic table and returns a comma-delimited list of all databases for which entities have been imported. This API is for cases where you want the names of all databases for which entities have been imported, rather than just the connected databases.

Location: af\app\afgenmgrp.i

Parameters:

INPUT p1OnlyUseCache AS LOGICAL

This parameter is redundant.

OUTPUT pcDBList AS CHARACTER

The list of databases for which entities have been imported.

INPUT-OUTPUT pcAdditionalInfo AS CHARACTER

This parameter is for future use. It serves no function in the current version.

Notes: None

Examples: See the main block in af\app\afgengtdbfrimpentp.p.

4.3.8 getDBVersion

This procedure returns the database version number from the database sequence initial value. The sequence name must match the naming convention, seq_<dblogical>_dbversion, for example, seq_icfdb_dbversion. If a sequence with this name is not found for a passed-in database, the version number is returned as 000000.

The ERwin model has a UDP, called DBVersion, that causes this sequence to be created and updated with the correct value when a delta file is generated.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcLogicalNames AS CHARACTER

A comma-delimited list of logical database names.

OUTPUT pcVersions AS CHARACTER

The database version as a string, for example, 020005.

Notes: This is used by the install and the help about window.

Examples: See the main block in af\app\afappingp.p.

4.3.9 getDumpName

This procedure checks if a specified table exists in a database.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcQuery AS CHARACTER

The database name and table name, delimited by a pipe, |.

OUTPUT pcTableDumpName AS CHARACTER

Notes: None

Examples: See the getTableDumpName function in af\app\afgenmgrp.i.

4.3.10 **getEntityCacheBuffer**

This function returns the buffer handle of the temp-table used to store the cached entity mnemonic buffer.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcEntity AS CHARACTER

INPUT pcTableName AS CHARACTER

Returns: HANDLE

Notes: None

Examples: See the deletePostTransValidate procedure in af\obj2\gsmcmlogcp.p.

4.3.11 **getEntityDescription**

This procedure returns the reference number, -code parameter, or similar description field for an entity.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcEntityMnemonic AS CHARACTER

INPUT pdEntityObj AS DECIMAL

INPUT pcFieldName AS CHARACTER

OUTPUT pcEntityDescriptor AS CHARACTER

Notes: None

Examples: See the getOEMCode function in af\app\afgenmgrp.i.

4.3.12 **getEntityDetail**

This procedure returns information about an entity.

Location: af\app\afgenmnggrp.i

Parameters:

INPUT pcEntity AS CHARACTER

The entity mnemonic.

OUTPUT pcEntityFields AS CHARACTER

A CHR(1)-delimited list of the entity mnemonic field names.

OUTPUT pcEntityValues AS CHARACTER

A CHR(1)-delimited list of the values of the entity fields.

Returns: See Notes.

Notes: This function uses the cached gsc_entity_mnemonic Repository table. If no record exists in the cache, then the return values are blank.

Examples: See the initializeObject procedure in af\obj2\gsmcmfullto.w.

4.3.13 **getEntityDisplayField**

This procedure returns the value defined in the entity_description_field using the pcOwningValue passed in for the fields in either entity_object_field or entity_key_field.

Location: af\app\afgenmnggrp.i

Parameters:

INPUT pcEntityMnemonic AS CHARACTER

INPUT pcOwningValue AS CHARACTER

OUTPUT pcEntityLabel AS CHARACTER

OUTPUT pcEntityDescriptor AS CHARACTER

Notes: None

Examples: See the entityUpdateDetail procedure in af\obj2\gsmcmvieww.w.

4.3.14 **getEntityExists**

This procedure checks that the specified record exists.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcTableName AS CHARACTER

INPUT pdTableObj AS DECIMAL

OUTPUT plExists AS LOGICAL

OUTPUT pcRejection AS CHARACTER

Notes: None

Examples: See the main block in af\app\afgengtentexstp.p.

4.3.15 **getEntityFieldCacheBuffer**

This function returns the buffer handle of the temp-table used to store the cached entity display field buffer.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcEntity AS CHARACTER

INPUT pcTableName AS CHARACTER

Returns: HANDLE

Notes: None

Examples: See the main block in ry\inc\rygenogtji.i.

4.3.16 getEntityTableName

This procedure returns the table name based on an entity.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcEntityMnemonic AS CHARACTER

INPUT pcLogicalDBName AS CHARACTER

OUTPUT pcEntityTablename AS CHARACTER

Notes: This function runs only on the AppServer or a DB-aware client.

Examples: See the buildQuery procedure in af\app\secgetdata.p.

4.3.17 getHighKey

This function returns the character key with the highest sort rank for the given collation.

You can use this character to make a high key. For example, to search all customer names from “A” to “B”, inclusive, you can use the following:

```
ASSIGN low = "A"
      high = "B" + getHighKey(session:cpcoll).
FOR EACH customer WHERE custname >= low AND custname <= high.
```

If you are preparing to use the COMPARE statement or COLLATE phrase with a specified collation table name, then you can supply that table name to this function to get the high key for that collation.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcCollationTableName AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the applyFilter procedure in af\sup2\afsdofiltw.w.

4.3.18 **getInternalEntries**

This function passes back internal entries of an SDO, since internal entries cannot be accessed for remote proxy procedures.

Location: af\app\afgenmgrp.i

Parameters: None

Returns: CHARACTER

Notes: None

Examples: See the getInternalEntryExists in af\app\afsesmgrp.i.

4.3.19 **getKeyField**

This function retrieves the key field from a cached record in the ttEntityMnemonic temp-table.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcEntityMnemonic AS CHARACTER

 The entity mnemonic to look up on the temp-table.

Returns: CHARACTER

Notes: None

Examples: See the getTableInfo procedure in af\app\afgenmgrp.i.

4.3.20 **getLanguageText**

This procedure retrieves the language text in the specified language. If a specific text TLA is not specified, then all values for the specified language are returned as a pipe (|) delimited list. If the language is not specified, then the current login language is used.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcCategoryType AS CHARACTER

INPUT pcCategoryGroup AS CHARACTER

INPUT pcSubGroup AS CHARACTER

INPUT pcTextTla AS CHARACTER

If no value is passed in for this optional parameter, all are used.

INPUT pdLanguageObj AS DECIMAL

If no value is passed in for this optional parameter, defaults to login language.

INPUT pdOwningObj AS DECIMAL

An optional parameter, might not be applicable.

INPUT pcSubstitute AS CHARACTER

An optional parameter, a pipe (|) delimited list when used.

OUTPUT pcLanguageText AS CHARACTER

If multiple values, list is pipe (|) delimited.

OUTPUT pcFileName AS CHARACTER

If multiple values, list is pipe (|) delimited.

Notes:

- If substitutions are passed in, the substitutions are made everywhere a placeholder is found in the text, for example, {1}. They should be passed in as pipe (|) delimited lists.
- If the contents are empty, then the unknown value, ?, is returned.
- If the filename is empty, then the unknown value, ?, is returned.

Examples: See the main block in af\app\afgengtngtxt.p.

4.3.21 getNextSequenceValue

This function returns the next sequence value for a given sequence.

Location: af\app\afgenmnggrp.i

Parameters:

INPUT pdCompanyObj AS DECIMAL

INPUT pcEntityMnemonic AS CHARACTER

INPUT pcSequenceTLA AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the beginTransactionValidate procedure in af\obj2\gsmmi follow.

4.3.22 getObjField

This function retrieves the field that serves as the object (xxx_obj) field from a cached record in the ttEntityMnemonic temp-table.

The function uses the following rules:

1. If no record exists in the ttEntityMnemonic table or the value of ttEntityMnemonicCache.Table_has_object_field is "NO", it returns "" (blank).
2. If ttEntityMnemonicCache.Entity_object_field is not "" (blank), it returns that value.

Location: af\app\afgenmnggrp.i

Parameters:

INPUT pcEntityMnemonic AS CHARACTER

The entity mnemonic to look up on the temp-table.

Returns: CHARACTER

Notes: None

Examples: See the checkIfOverlaps procedure in af\app\afgenmnggrp.i.

4.3.23 getOEMCode

This function returns the reference number, -code, or similar key field.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcEntityMnemonic AS CHARACTER

INPUT pdEntityObj AS DECIMAL

Returns: CHARACTER

Notes: Use the getOEMDescription function for a name or description field.

4.3.24 getOEMDescription

This function returns a name or similar description field.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcEntityMnemonic AS CHARACTER

INPUT pdEntityObj AS DECIMAL

Returns: CHARACTER

Notes: Use the getOEMCode function for a reference number, -code, and similar values.

4.3.25 getPropertyFromList

This function returns the value of a specified property in a specified property list.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcPropertyList AS CHARACTER

INPUT pcPropertyName AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the constructObject procedure in af\obj2\gsmcmvieww.w.

4.3.26 **getRecordCheckAudit**

This procedure checks whether an audit record exists for a specific record.

Location: af\app\afgenmnggrp.i

Parameters:

INPUT pcEntityMnemonic AS CHARACTER

 The entity mnemonic.

INPUT pcEntityObjField AS CHARACTER

 The entity mnemonic object field as a string.

INPUT pcEntityObjValue AS CHARACTER

 The entity mnemonic object field value as a string.

OUTPUT plRowAuditExist AS LOGICAL

 If an audit record exists for the record, this is set to YES.

Notes: None

4.3.27 **getRecordCheckComment**

This procedure checks whether a comment record exists for a specific record.

Location: af\app\afgenmnggrp.i

Parameters:

INPUT pcEntityMnemonic AS CHARACTER

 The entity mnemonic.

INPUT pcEntityObjField AS CHARACTER

 The entity mnemonic object field as a string.

INPUT pcEntityObjValue AS CHARACTER

 The entity mnemonic object field value as a string.

OUTPUT p1RowCommentExist AS LOGICAL

If an audit record exists for the record, this is set to YES.

OUTPUT pcRowCommentAuto AS CHARACTER

Notes: None

4.3.28 getRecordDetail

This procedure returns the details of the first record found for a specified query.

Location: af\app\afgenmnggrp.i

Parameters:

INPUT pcQuery AS CHARACTER

A valid FOR EACH query.

OUTPUT pcFieldList AS CHARACTER

A CHR(3)-delimited list of all fields in the query and their values.

Notes: None

Examples: See the setChangedState procedure in af\obj2\gsmcmvieww.w.

4.3.29 getRecordUserProp

This procedure determines if audit or comment records exist for a user.

Location: af\app\afgenmnggrp.i

Parameters:

INPUT pcEntityMnemonic AS CHARACTER

INPUT pcEntityFields AS CHARACTER

INPUT pcEntityObjValue AS CHARACTER

OUTPUT pcRowUserProp AS CHARACTER

Notes: None

Examples: See the main block in af\app\afgengtrecusrprpp.p.

4.3.30 **getSequenceConfirmation**

This function checks whether a specified sequence is valid.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pdCompanyObj AS DECIMAL

INPUT pcEntityMnemonic AS CHARACTER

INPUT pcSequenceTLA AS CHARACTER

Returns: LOGICAL

Notes: None

4.3.31 **getSequenceExist**

This procedure determines whether a sequence or reference exists.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pdCompanyObj AS DECIMAL

INPUT pcEntityMnemonic AS CHARACTER

INPUT pcSequenceTLA AS CHARACTER

OUTPUT plSuccess AS LOGICAL

Notes: None

Examples: See the getSequenceConfirmation function in af\app\afgenmgrp.i.

4.3.32 **getSequenceMask**

This procedure builds a format mask for a sequence. This format mask is used to format integers. It calculates the number of digits to display based on the number between the brackets, []. If there is no value, it defaults to eight characters.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcSequenceFormat AS CHARACTER

OUTPUT pcQuantityIndicator AS CHARACTER

OUTPUT pcSequenceMask AS CHARACTER

Notes: None

Examples: See the getSequenceValue procedure in af\app\afgenmgrp.i.

4.3.33 getSiteNumber

This procedure determines the current Repository database site number from the ICFDB database sequence.

Location: af\app\afgenmgrp.i

Parameters:

OUTPUT piSite AS INTEGER

The current site number.

Notes: None

Examples: See the getSequenceValue procedure in af\app\afgenmgrp.i.

4.3.34 getStatusObj

This function returns a status object number to a calling procedure.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcCategoryType AS CHARACTER

INPUT pcCategoryGroup AS CHARACTER

INPUT pcCategorySubGroup AS CHARACTER

Returns: DECIMAL

Notes: This is a wrapper function that returns the object number. The getStatusRecord procedure maintains the cached status records.

4.3.35 **getStatusShortDesc**

This function returns a status short description to a calling procedure.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pdStatusObj AS DECIMAL

Returns: CHARACTER

Notes: This is a wrapper function that returns the object number. The
getStatusRecord procedure maintains the cached status records.

4.3.36 **getStatusTLA**

This function returns the status TLA to a calling procedure.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pdStatusObj AS DECIMAL

Returns: CHARACTER

Notes: This is a wrapper function that returns the object number. The
getStatusRecord procedure maintains the cached status records.

4.3.37 **getTableDumpName**

This function checks if a specified table exists in a database.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcQuery AS CHARACTER

 The database name and table name as a pipe (|) delimited pair.

Returns: CHARACTER

Notes: None

Examples: See the getTableInfo procedure in af\app\afgenmgrp.i .

4.3.38 **getTableInfo**

This procedure returns a CHR(4)-delimited list of the following information about the updatable table in the supplied data source:

- Entry 1 – The owning entity mnemonic.
- Entry 2 – The table name.
- Entry 3 – The table's key field. This is not necessarily the object (_obj) field.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcUpdatableTable AS CHARACTER

OUTPUT pcReturnValue AS CHARACTER

Notes: None

Examples: See the getUpdatableTableInfo function in af\app\afgenmgrp.i.

4.3.39 **getTableInfoObj**

This procedure returns information about the updatable table in the supplied data source in a CHR(4) delimited list of the following order: owning entity mnemonic, table name, obj field of table.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcUpdatableTable AS CHARACTER

OUTPUT pcReturnValue AS CHARACTER

Notes: This procedure differs from getTableInfo only in that it returns an obj field for the third entry in the return value.

Examples: See the getUpdatableTableInfoObj function in af\app\afgenmgrp.i.

4.3.40 **getUpdatableTableInfo**

This function returns a CHR(4)-delimited list of the following information about the updatable table in the supplied data source:

- **Entry 1**—The owning entity mnemonic.
- **Entry 2**—The table name.
- **Entry 3**—The table's key field. This is not necessarily the object (_obj) field.

Location: af\app\afgenmnggrp.i

Parameters:

INPUT phDataSource AS WIDGET-HANDLE

 The handle of an SDO for which you want to retrieve data.

Returns: CHARACTER

Notes: None

Examples: See the getComboQuery procedure in af\obj2\gsmcad3sfv.w.

4.3.41 **getUpdatableTableInfoObj**

This function returns information about the updatable table in the supplied data source in a CHR(4) delimited list of the following order: owning entity mnemonic, table name, obj field of table.

Location: af\app\afgenmnggrp.i

Parameters:

INPUT phDataSource AS WIDGET-HANDLE

 The handle of an SDO that you want to retrieve data for.

Returns: CHARACTER

Notes: This function only differs from getUpdatableTableInfo in that it returns an obj field for the third entry in the list.

4.3.42 **getUserSourceLanguage**

This procedure returns the source language for a user.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pdUserObj AS DECIMAL

OUTPUT pdSourceLanguageObj AS DECIMAL

Notes: This function only runs on the AppServer or a DB-aware client.

Examples: See the addRecord procedure in af\obj2\gsmmiview.w.

4.3.43 **haveOutstandingUpdates**

This function checks the ryt_dbupdate_status record for any outstanding updates.

Location: af\app\afgenmgrp.i

Parameters: None

Returns: LOGICAL

Notes: None

Examples: See the initializeSession in af\app\afxm1cfgp.p.

4.3.44 **listLookup**

This function returns the value of an element in pcTarget as indicated by the position for pcElement in pcSource.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcElement AS CHARACTER

INPUT pcSource AS CHARACTER

INPUT pcTarget AS CHARACTER

INPUT pcDelimiter AS CHARACTER

Defaults to commas.

Returns: CHARACTER

Notes: None

4.3.45 **plipShutdown**

This procedure is part of the standard Progress Dynamics Manager template. It runs on close of the procedure.

Location: af\app\afgenmnggrp.i

Parameters: None

Notes: This procedure currently contains no active code.

4.3.46 **refreshMnemonicsCache**

This procedure loads the latest set of entity mnemonic and entity display field records into the cache.

Location: af\app\afgenmnggrp.i

Parameters: None

Notes: None

Examples: See the validateEntityMnemonic procedure in af\app\afgenmnggrp.i.

4.3.47 **setPropertyValuesInList**

This function finds a property in the property list. Then, depending on the specified action, the function adds or replaces the property with the specified property value.

Location: af\app\afgenmnggrp.i

Parameters:

INPUT pcPropertyList AS CHARACTER

INPUT pcPropertyName AS CHARACTER

INPUT pcPropertyValue AS CHARACTER

INPUT pcAction AS CHARACTER

Should be either "ADD":U or "REPLACE":U. If blank or unknown, it defaults to REPLACE.

Returns: CHARACTER

Notes: None

Examples: See the updateProperties procedure in af\obj2\gsmcmlogcp.p.

4.3.48 **setWidgetAttribute**

This function sets an attribute value for the passed-in widget handle. If desired, the CREATE CALL... statement can be made externally, this will improve performance.

Location: af\app\afgenmgrp.i

Parameters:

INPUT phWidget AS HANDLE

INPUT pcAttributeName AS CHARACTER

INPUT pcAttributeValue AS CHARACTER

INPUT pcAttributeDataType AS CHARACTER

INPUT phExternalCall AS HANDLE

Returns: LOGICAL

Notes: None

4.3.49 **updateTableViaSDO**

This procedure updates the database through the relevant SDO.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcSdoName AS CHARACTER

 The filename of the SDO used in the update.

INPUT pcSdoDescription AS CHARACTER

 The description of the SDO.

INPUT pcExtraProperties AS CHARACTER

 A CHR(3)-delimited string of additional properties to set in the SDO (see Notes).

INPUT pcUserProperties AS CHARACTER

 A CHR(3)-delimited string of additional user properties to set in the SDO (see Notes).

INPUT-OUTPUT TABLE-HANDLE phRowObjUpd

 The rowObjUpd table containing records for update.

Returns: See Notes

Notes:

- Because this procedure always attempts to run the SDO on the ICF AppServer, no distinction is made between the client and server when running this procedure. If the calling procedure is already running on the server, the value of the gshAstraAppserver handle is that of the SESSION to avoid problems.
- This procedure starts the SDO, updates the database, and destroys the SDO after any updates. Any errors are reported back to the calling procedure using the RETURN-VALUE.
- The ERROR-STATUS error condition is raised; you should not run this procedure with the NO-ERROR option.
- The table handle is returned to the calling procedure in case there are further actions to be performed on it, or post-update data is required. For example, a status that is determined in the SDO.
- The extra properties string is a CHR(3)-delimited label, value string. These properties are set in the SDO after initialization.

Examples: See the deletePreTransValidate procedure in ry\obj\rycoillog3p.p.

4.3.50 validateEntityMnemonic

This procedure validates an entity mnemonic. It runs against the Repository's gsc_entity_mnemonic table first. If the entity mnemonic is found, then the procedure forces a refresh of the cache. If the entity mnemonic cannot be found in the database or the cache, then it returns a blank value. This procedure does not create entity mnemonic records.

Location: af\app\afgenmgrp.i

Parameters:

INPUT pcEntityMnemonic AS CHARACTER

The entity mnemonic to be checked.

OUTPUT pcResult AS CHARACTER

The results of the validation.

Notes: None

Localization Manager

The Localization Manager handles run-time translations of your user interface (UI).

This chapter covers the following subjects:

- [Overview](#)
- [Temp-tables used by Localization Manager](#)
- [Localization Manager APIs](#)

5.1 Overview

The Localization Manager supports the run-time translation of all the elements in the context of an application screen. This includes such items as window titles, labels, and messages. This aids your deployment of an application that needs to run in many different languages.

In a distributed environment, the Localization Manager is started on both the client and server sides. There is a wrapper program for each side that either sets the global variable, `server-side`, to YES for the server-side version or to NO for the client-side version. Both programs then include the main code of the manager from the `af\app\aftrnmngrp.i` include file. The differences in how the manager operates on each side are triggered by checking the value of the `server-side` variable.

The Localization Manager's handle is stored in the global shared variable, `gshTranslationManager`.

[Table 5-1](#) shows the files that contain the Localization Manager's code.

Table 5-1: Localization Manager files

Type	Path
Client-side wrapper	af\app\aftrnc1ntp.p
Server-side wrapper	af\app\aftrnsrvrp.p
Main code	af\app\aftrnmngrp.i
Included files	adm\method\attribut.i adm2\tttranslate.i af\app\aftrnbldtrncp.i af\app\aftrngetmtrnp.i af\app\aftrnupdmtrnp.i af\app\aftttranslation.i af\sup2\afcheckerr.i af\sup2\afglobals.i

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in the [Progress Dynamics Programming Handbook](#).

5.2 Temp-tables used by Localization Manager

The Localization Manager uses the temp-table described in [Table 5–2](#), a copy of the Repository `gsm_translation` table, to cache information about translations on the local client.

Table 5–2: Temp-tables defined in `aflaplafttttranslation.i`

Temp-table	Fields (data types)
ttTranslation	translation_obj (Decimal) language_obj (Decimal) object_filename (Character) widget_type (Character) widget_name (Character) widget_entry (Integer) original_label (Character) translation_label (Character) original_tooltip (Character) translation_tooltip (Character) source_language_obj (Decimal)

The Localization Manager also uses the temp-table described in [Table 5–3](#) to store information about translations.

Table 5–3: Temp-tables defined in `adm2\ttttranslate.i`

Temp-table	Fields (data types)
ttTranslate	dLanguageObj (Decimal) dSourceLanguageObj (Decimal) cLanguageName (Character) cObjectName (Character) lGlobal (Logical) lExtractedGlobal (Logical) cWidgetType (Character) cWidgetName (Character) hWidgetHandle (Handle) iWidgetEntry (Integer) lDelete (Logical) cTranslatedLabel (Character) cOriginalLabel (Character) cTranslatedTooltip (Character) cOriginalTooltip (Character)

5.3 Localization Manager APIs

This section lists the APIs that you can use with the Localization Manager.

5.3.1 buildClientCache

This procedure loads the local client cache temp-table, ttTranslation, for the current logged language.

Location: af\app\aftrnmngrp.i

Parameters:

INPUT pdLanguageObj AS DECIMAL

The language obj, or 0 for all.

Notes:

- Clear out the temp-table before running this procedure.
- If a language is specified, this procedure is run at application startup for maximum performance.

Examples: See the ICFCFM_Login procedure in icf\icfstart.p.

5.3.2 buildWidgetTable

This procedure builds a temp-table of widgets that might require translation by walking the widget tree for the given frame handle. It does not actually do the translation. The temp-table must then be passed to the translateWidgetTable procedure for subsequent translation.

If the window handle is known and passed in, then an entry is created in the temp-table for the window translation. If the object name is passed in as blank, then only translations for the objects in the window are retrieved.

Location: af\app\aftrnmngrp.i

Parameters:

INPUT pcObjectName AS CHARACTER

The object name, with path stripped if passed in. Blank = all.

INPUT pdLanguageObj AS DECIMAL

The language object number, or 0 for the login language.

INPUT phWindow AS HANDLE

The window handle (CURRENT-WINDOW), if title translation required.

INPUT phFrame AS HANDLE

The frame handle (FRAME {&FRAME-NAME}:HANDLE).

OUTPUT TABLE FOR ttTranslate

The temp-table of widgets for translation.

Notes: Since the frame handle is not available on the server, this procedure is only valid when run on the client.

Examples: See the buildTranslations procedure in af\cod2\aftem1ognw.w.

5.3.3 clearClientCache

This procedure ensures up-to-date information is retrieved from the database by emptying the client cache temp-table, ttTranslation. This procedure might be called when Repository maintenance programs have run. By using this procedure, you can avoid logging off and starting a new session to use new Repository data settings.

Location: af\app\aftrnmgrp.i

Parameters: None

Notes: None

5.3.4 **getTranslation**

This procedure checks if any translations exist for the passed-in details.

If a language is not passed in, the procedure uses the specified login language. If the object passed in is a specific object, the procedure first looks for a translation for that specific object. If none exists, it looks for a translation for all objects (blank object name). The procedure can handle title, browse, fill-in, radio set, text, button, toggle-box, combo-box, slider, and editor widgets.

Location: af\app\aftrnmngrp.i

Parameters:

INPUT pdLanguageObj AS DECIMAL

 The language object number, or 0 for the login language.

INPUT pcObjectName AS CHARACTER

 The object name for which to get the translation, or blank for all.

INPUT pcWidgetType AS CHARACTER

 The widget type.

INPUT pcWidgetName AS CHARACTER

 The widget name (LABEL) or, if it is a text widget, the text to translate.

INPUT pcWidgetEntry AS INTEGER

 The widget entry, used for radio sets and similar widgets

OUTPUT pcOriginalLabel AS CHARACTER

 The original untranslated label.

OUTPUT pcTranslatedLabel AS CHARACTER

 The translated label.

OUTPUT pcOriginalTooltip AS CHARACTER

 The original untranslated ToolTip.

OUTPUT pcTranslatedTooltip AS CHARACTER

 The translated ToolTip.

Notes:

- A window title translation is a special case that uses “title” as the value for both widget type and widget name.
- Column label translations use a widget type of “browse” and a widget name of the column label.
- ToolTip translations are ignored for text widgets.
- For a procedure running client-side, if the cachedTranslationsOnly property is set to YES in the Session Manager, then the database is not checked for translations that are not in the cache.

Examples: See the translatePhrase function in af\app\aftrnmgrp.i.

5.3.5 multiTranslation

This procedure performs multiple translations in one pass. It stores the translation results in a temp-table, ttTranslate, for later use by the caller. The generated temp-table contains the widget type and handle for all the translated objects, so using the translated values is a simple process.

When the flag is set to translate all languages, extra records are created in the temp-table for each available language. The temp-table must initially contain entries with a language obj of 0. These entries are deleted following the translation into each language.

Location: af\app\aftrnmgrp.i

Parameters:

INPUT p1AllLanguages AS LOGICAL

Set to YES to translate for all languages.

INPUT-OUTPUT TABLE FOR ttTranslate

The temp-table of translations.

Notes:

- The all languages option works by accessing the database. It cannot run off cached information.
- Dynamic toolbars only support global translations.
- Window titles and tabs only support specific object translations.

Examples: See the translateWidgets procedure in af\app\afsesmgrp.i.

5.3.6 plipShutdown

This procedure is a part of the standard Progress Dynamics Manager template. It runs on close of the procedure.

Location: af\app\aftrnmngrp.i

Parameters: None

Notes: This procedure currently contains no active code.

5.3.7 receiveCacheClient

This procedure sets the global variable, gdCachedLanguageObj, to the current language.

Location: af\app\aftrnmngrp.i

Parameters:

INPUT TABLE FOR ttTranslation

INPUT pdLanguageObj AS DECIMAL

Notes: None

Examples: See the loginCacheAfter procedure in af\app\afsesmngpr.i.

5.3.8 translatePhrase

This function translates a text phrase into the given language. The function returns the translated text. If no translation is found, it returns the original text.

Location: af\app\aftrnmngrp.i

Parameters:

INPUT pcText AS CHARACTER

The text to translate.

INPUT pdLanguageObj AS DECIMAL

The language into which to translate the phrase, defaulting to the login language.

Returns: Character

Notes: None

Examples: See the main block in af\app\afmessagep.p.

5.3.9 translateWidgetTable

This procedure translates the widgets temp-table, ttTranslate, from the buildWidgetTable procedure. The procedure uses the multiTranslation procedure, if it has not already been run.

The “already translated” flag allows the multiTranslation procedure to run prior to this procedure so that all languages can be translated in one pass if required. The table of widgets can therefore contain entries for each language. The procedure uses the entries for the passed in language or the login language. The temp-table might only have entries for the login language, in which case the language obj is 0.

Location: af\app\aftrnmgrp.i

Parameters:

INPUT p1Translated AS LOGICAL

If the multiTranslation procedure has run, set to YES.

INPUT pdLanguageObj AS DECIMAL

The translation language object number to use. Use 0 for the login language.

INPUT TABLE FOR ttTranslate

The temp-table of widgets to translate.

Notes: This procedure can only run on the client, because the handles are not valid on the server.

Examples: See the doTranslations procedure in af\cod2\aftemcpasw.w.

5.3.10 updateTranslations

This procedure updates translations to the database and the client cache, if appropriate.

Location: af\app\aftrnmgrp.i

Parameters:

INPUT TABLE FOR ttTranslate

The temp-table of translations.

Notes: None

Examples: See the updateRecord procedure in ry\obj\rydyntranv.w.

Profile Manager

The Profile Manager handles access to information on user preferences for the Progress Dynamics framework.

This chapter covers the following subjects:

- [Overview](#)
- [Temp-tables used by Profile Manager](#)
- [Profile Manager APIs](#)

6.1 Overview

The Profile Manager encapsulates all user profile access supported by the framework. User profile data includes such run-time information as the user’s preferences for window positions and sizes, browser filter settings, and report filter settings. It also includes system wide information, such as ToolTips enabled and similar UI options. The Profile Manager stores all of these settings by user. The user can decided to save the settings only for the current session or save them permanently in the Repository for consistency between client sessions.

In a distributed environment, the Profile Manager is started on both the client and server sides. There is a wrapper program for each side that either sets the global variable, `server-side`, to YES for the server-side version or to NO for the client-side version. Both programs then include the main code of the manager from the `af\app\afpromngrp.i` include file. The differences in how the manager operates on each side are triggered by checking the value of the `server-side` variable.

The Profile Manager’s handle is stored in the global shared variable, `gshProfileManager`.

[Table 6–1](#) shows the files that contain the Profile Manager’s code.

Table 6–1: Profile Manager files

Type	Path
Client-side wrapper	af\sup2\afproc1ntp.p
Server-side wrapper	af\app\afprosrvrp.p
Main code	af\app\afpromngrp.i
Included files	adm\method\attribut.i af\app\afbldclcp.p af\app\afproupcadbp.i af\app\afttprofiledata.i af\sup2\afcheckerr.i af\sup2\aferrortxt.i af\sup2\afglobals.i

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in the *Progress Dynamics Programming Handbook*.

6.2 Temp-tables used by Profile Manager

The Profile Manager uses the temp-table described in [Table 6–2](#), a copy of the Repository `gsm_profile_data` table with some extra fields, to cache the user’s profile information.

Table 6–2: Temp-table defined in `aflaplafttprofiledata.i`

Temp-table	Fields (data types)
ttProfileData	user_obj (Decimal) profile_type_obj (Decimal) profile_code_obj (Decimal) profile_data_key (Character) context_id (Character) profile_data_value (Character) profile_data_obj (Decimal) cProfileTypeCode (Character) cProfileCode (Character) cAction (Character)

6.3 Profile Manager APIs

This section lists the APIs that you can use with the Profile Manager.

6.3.1 buildClientCache

This procedure loads the local client cache temp-table for the current logged-in user. It loads only permanent profile data. Permanent data has a context id value of blank. A non-blank context id indicates that the profile data is specific to a given session. The procedure only loads data into the cache for profiles that are client-side-only data.

Location: af\app\afpromngrp.i

Parameters:

INPUT pcProfileTypeCodes AS CHARACTER

A comma-delimited list of specific profile type codes (blank = all).

Notes:

- This procedure clears out the temp-table before reading profile data from the Repository.
- This procedure is run at application start-up for maximum performance.
- If this procedure runs after application start-up, it clears out the existing temp-table.

Examples: See the relogon procedure in af\app\afsesmgrp.i.

6.3.2 checkProfileDataExists

This procedure checks whether profile data exists for the current logged-in user for the specified profile type, code, or key. This procedure supports the passing of partial information, for example, only the profile type, or the type and code.

Location: af\app\afpromngrp.i

Parameters:

INPUT pcProfileTypeCode AS CHARACTER

The profile type code.

INPUT pcProfileCode AS CHARACTER

The profile code.

INPUT pcProfileDataKey AS CHARACTER

The profile data key.

INPUT p1CheckPermanentOnly AS LOGICAL

Check permanent data only flag, default = YES.

INPUT p1CheckCacheOnly AS LOGICAL

Check client cache only, default = NO.

OUTPUT p1Exists AS LOGICAL

Does the profile data exist.

Notes:

- If the permanent data only flag is set to NO and there are database entries that exist for the session only, then the procedure returns TRUE.
- If the check client only cache is set to YES, then the procedure checks the temp-table, rather than checking both the temp-table and the database.

Examples: See the initializeObject procedure in ry\obj\rydynprf1v.w.

6.3.3 clearClientCache

This procedure ensures up-to-date information is retrieved from the database by emptying the client cache temp-tables. This might be called when maintenance programs have been run that change profile data. Using this procedure enables you to avoid logging off and starting a new session in order to use the new profile data settings.

Location: af\app\afpromngrp.i

Parameters: None

Notes: When this procedure runs, any profile changes made within the session and not already committed to the database are lost.

Examples: See the relogon procedure in af\app\afsesmgrp.i.

6.3.4 deleteSessionProfile

This procedure deletes session-specific profile data. It is run from `dynamics\as_disconnect.p` when a client disconnects from an agent.

User profile data can be permanent or scoped to a specific session. Permanent data is stored between sessions and has a blank `context_id`. Session-specific data is not stored between sessions and has a non-blank `context_id`.

Location: `af\app\afpromngrp.i`

Parameters: None

Notes: This procedure must use the actual `SESSION:SERVER-CONNECTION-ID` and not the `gscSessionId`. The `gscSessionId` might have been set to null by the time this procedure runs.

Examples: See the main block in `icf\as_disconnect.p`.

6.3.5 getProfileData

This procedure returns profile data value for a specified profile for the current user for the current session. If the procedure is running on the client side, it first looks in client cache for the profile data value. If the profile data value is not there, the procedure looks in the database.

If a `rowid` is passed in, then the `rowid` is used to find the record. For a client-only profile type, the `rowid` for the temp-table is used. Otherwise, the `rowid` of the database record is used. A profile type must be passed in to determine whether the `rowid` is from the temp-table or the database. If a `rowid` is passed in, it takes precedence over any other input parameters.

If the next flag is set to YES, then a FIND NEXT is done to retrieve the record after the passed-in record. If the record is cached on the client, the `rowid` of the temp-table is returned. Otherwise, the `rowid` of the database record is returned. The `rowid` is useful when reading through sets of profile data using the next functionality.

Location: `af\app\afpromngrp.i`

Parameters:

INPUT `pcProfileTypeCode` AS CHARACTER

The profile type code.

INPUT `pcProfileCode` AS CHARACTER

The profile code.

INPUT pcProfileDataKey AS CHARACTER

The profile data key.

INPUT plNextRecordFlag AS LOGICAL

The next flag, YES = get next value.

INPUT-OUTPUT prRowid AS ROWID

The rowid of the profile data.

OUTPUT pcProfileDataValue AS CHARACTER

The profile data value.

Notes: Always check for session data first, and then for permanent data.

Examples: See the getUserSourceLanguage procedure in af\app\afgenmgrp.i.

6.3.6 getProfileTTHandle

This function returns the handle of the ttProfileData temp-table.

Location: af\app\afpromngrp.i

Parameters: None

Returns: HANDLE

Notes: None

Examples: See the resetSesPerFlag procedure in af\sup2\afsdofiltw.w.

6.3.7 plipShutdown

This procedure is a part of the standard Progress Dynamics Manager template. On close of the procedure, if the session is not a remote session, it runs the updateCacheToDb procedure.

Location: af\app\afpromngrp.i

Parameters: None

Notes: None

6.3.8 receiveProfileCache

This procedure takes an existing set of profiling records that have been populated, and populates them with the profile information set on the Repository.

This procedure supplements the profile cache with information from an external source. The login process uses it to add to the profile cache. It is also used when launching a container.

Location: af\app\afpromngrp.i

Parameters:

INPUT TABLE FOR ttProfileData APPEND.

Notes: This procedure should only be run on the server.

Examples: See the loginCacheAfter procedure in af\app\afsesmgrp.i.

6.3.9 setProfileData

This procedure sets profile data values for the specified profile for the current user in the current session. You can use this procedure to set values for or to delete a single record. Alternately, by using blank as the value for the pcProfileCode and pcProfileDataKey parameters, you can set a value for all records or delete all records.

If a rowid is passed in, then the record to set is found using the rowid, rather than using the individual profile fields. The record must already exist to be set in this way. For a client-only profile type, the rowid of the temp-table is used. Otherwise, the rowid of the database record is used. A profile type is passed in with the rowid to indicate whether it is from the temp-table of the database. If a rowid is passed in, it takes precedence over any other input parameters.

When the client cache is updated, the build cache external procedure must be run to fully build the temp-table with entries for this profile type for the current user. This ensures that entries are either updated fully into the cache or the database. This behavior also ensures that a record exists for the user for every profile code in the profile type. In this way, all the information needed to create new records in the cache is available, for example, the object numbers.

Location: af\app\afpromngrp.i

Parameters:

INPUT pcProfileTypeCode AS CHARACTER

The profile type code.

INPUT pcProfileCode AS CHARACTER

The profile code, (blank = all).

INPUT pcProfileDataKey AS CHARACTER

The profile data key, (blank = all).

INPUT prRowid AS ROWID

The rowid of profile data, if known.

INPUT pcProfileDataValue AS CHARACTER

The profile data value.

INPUT plDeleteFlag AS LOGICAL

The delete flag (YES = delete profile data).

INPUT pcSaveFlag AS CHARACTER

The save flag (SES = Session only, PER = Permanent).

Notes:

- If the procedure runs on the client and the profile type is client-only, the procedure only updates the client cache with the new data value. The database is updated at session end, or can be manually updated using `updateCacheToDb`.
- If the procedure runs on the server or the profile type is server, the procedure updates the database immediately with the new profile data value.
- If the `pcSaveFlag` flag is set to session, the procedure stores the session ID in the context ID field. Leaving the field blank indicates a permanent value.
- If the `plDeleteFlag` flag is set to YES and client caching is valid, the procedure marks the data value for deletion in the temp-table. Otherwise, the data value is deleted directly from the database table.

Examples: See the `setUserProfile` procedure in `af\cod\afsvwizdw.w`.

6.3.10 updateCacheToDb

This procedure updates the modified details from the client cache temp-table into the database. It then resets the action flag on the updated temp-tables to NON to ensure they are not updated again.

This procedure is run by the plipShutdown procedure when the session ends. You might also run it manually at any stage to reflect maintenance changes.

Location: af\app\afpromngrp.i

Parameters:

INPUT pcProfileTypeCodes AS CHARACTER

A comma-delimited list of specific profile type codes, (blank = all).

Notes: None

Examples: See the relogin procedure in af\app\afsesmgrp.i.

Referential Integrity Manager

The Referential Integrity Manager handles data versioning and supports the reuse of unique keys.

This chapter covers the following subjects:

- [Overview](#)
- [Temp-tables used by Referential Integrity Manager](#)
- [Referential Integrity Manager APIs](#)

7.1 Overview

The Referential Integrity Manager provides support for data versioning and the reuse of unique keys. This manager does not record the modifications made to records in the Repository. It only records that the record has changed.

This manager should only be accessed through the versionData procedure.

In Progress Dynamics Version 1.1, there was a potential for clashes on unique key information. This could occur if a parent-level record was deleted and re-created after its original version had been deployed to a target database. To prevent this problem, the Referential Integrity Manager checks the Repository `gst_record_version` table for information about old keys. If you attempt to recreate a record with its old keys, the manager checks this table for a record of the old key information. If the record is found, the manager sets the object ID of the new record as the old object ID.

The Referential Integrity Manager’s handle is stored in the global shared variable, `gshRIManager`.

[Table 7–1](#) shows the files that contain the Referential Integrity Manager’s code.

Table 7–1: Referential Integrity Manager files

Type	Path
Client-side wrapper	NA
Server-side wrapper	NA
Main code	ry\app\ryrisrvrp.p
Included files	adm2\globals.i adm\method\attribut.i af\sup2\afcheckerr.i ry\app\ryplipkill.i ry\app\ryplipmain.i ry\app\ryplipsetu.i ry\app\ryplipshut.i

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in the *Progress Dynamics Programming Handbook*.

7.2 Temp-tables used by Referential Integrity Manager

The Referential Integrity Manager uses the temp-tables described in [Table 7–2](#) to store information about Repository tables and their relationships:

Table 7–2: Temp-tables defined in ry\app\ryrisrvrp.p (1 of 2)

Temp-table	Fields (data types)
ttEntity ¹	entity_mnemonic (Character) entity_mnemonic_short_desc (Character) entity_mnemonic_description (Character) auto_properform_strings (Logical) entity_mnemonic_label_prefix (Character) entity_mnemonic_obj (Character) entity_description_field (Character) entity_description_procedure (Character) entity_narration (Character) entity_object_field (Character) table_has_object_field (Logical) entity_key_field (Character) table_prefix_length (Integer) field_name_separator (Character) auditing_enabled (Logical) version_data (Logical) deploy_data (Logical) entity_dbname (Character) replicate_entity_mnemonic (Character) replicate_key (Character) scm_field_name (Character) reuse_deleted_keys (Logical)

Table 7–2: Temp-tables defined in ry\app\ryrisrvrp.p (2 of 2)

Temp-table	Fields (data types)
ttDSEntity ²	dataset_entity_obj (Decimal) deploy_dataset_obj (Decimal) entity_sequence (Integer) entity_mnemonic (Character) primary_entity (Logical) join_entity_mnemonic (Character) join_field_list (Character) filter_where_clause (Character) delete_related_records (Logical) overwrite_records (Logical) keep_own_site_data (Logical) use_relationship (Logical) relationship_obj (Decimal) deletion_action (Character) exclude_field_list (Character) cParentJoinString (Character) cChildJoinString (Character) cParentJoinFields (Character) cChildJoinFields (Character) cParentReplace (Character) cChildReplace (Character)

¹ This temp-table is a copy of the Repository's gsc_entity_mnemonic table.

² This temp-table is a copy of the Repository's gsc_dataset_entity table with some added fields.

7.3 Referential Integrity Manager APIs

This section lists the APIs that you can use with the Referential Integrity Manager.

7.3.1 versionData

This procedure coordinates versioning the data to be deployed.

Location: ry\app\ryrisrvrp.p

Parameters:

INPUT phBuffer AS HANDLE

INPUT pcFLA AS CHARACTER

INPUT pcAction AS CHARACTER

Notes: None

Examples: See the main block in af\app\afversionp.p.

Repository Managers

The Repository Managers handle the moving and caching of data between the Repository and a calling procedure.

This chapter covers the following subjects:

- [Overview](#)
- [Temp-tables used by Repository Managers](#)
- [Repository Manager APIs](#)
- [Repository Design Manager APIs](#)

8.1 Overview

The Repository Managers encapsulate all Repository-based access for building dynamic objects in the Progress Dynamics environment. They are used to access the data for the user interface (UI) and the objects that make up an application. The Managers coordinate the moving and caching of data between the Repository and a calling procedure. Examples of the Repository data transported include such items as object property (attribute) values, toolbar bands, and toolbar actions.

There are two Repository Managers, one that handles development tasks and another to handle runtime tasks. The Repository Manager handles data transport for running applications. It is designed to efficiently return object definitions to applications at runtime. The Repository Design Manager manages Repository access during application development tasks. It is designed to efficiently work with the AppBuilder, the Object Generator, and the other tools you use to create a Progress Dynamics application.

In a distributed environment, the Repository Managers are started on both the client and server sides. There is a wrapper program for each side that either sets the global variable, `server-side`, to YES for the server-side version or to NO for the client-side version. Each program then includes the main code of the manager from the appropriate include file. The differences in how the managers operate on each side are triggered by checking the value of the `server-side` variable.

The Repository Manager's handle is stored in the global shared variable, `gshRepositoryManager`.

You can retrieve the handle of the Repository Design Manager by running code like the following example:

```
ASSIGN hRepositoryDesignManager =  
DYNAMIC-FUNCTION("getManagerHandle":U, INPUT "RepositoryDesignManager":U).
```

Table 8–1 shows the files that contain the Repository Manager’s code.

Table 8–1: Repository Manager files

Type	Path
Client-side wrapper	ry\app\ryrepclntp
Server-side wrapper	ry\app\ryreprsvrp.p
Main code	ry\app\ryrepmgrp.i
Included files	adm2\calltables.i adm2\ttaction.i adm2\tttoolbar.i af\app\afdatatypi.i af\app\afttsecurityctrl.i af\app\afttttranslate.i af\sup2\aferrortxt.i af\sup2\afglobals.i ry\app\rydefrescd.i ry\app\rygetitemp.p ry\app\rygetmensp.p ry\app\rymenufunc.i ry\inc\getobjecti.i ry\inc\ryattstori.i ry\inc\ryrepatset.i ry\inc\ryrepdoori.i ry\inc\ryrepgeidi.i

Table 8–2 shows the files that contain the Repository Design Manager’s code.

Table 8–2: Repository Design Manager files

Type	Path
Client-side wrapper	ry\app\rydesc1ntp.p
Server-side wrapper	ry\app\rydessrvrp.p
Main code	ry\app\rydesmngp.i
Included files	adm2\calltables.i adm2\globals.i af\app\afdatatypi.i af\sup2\afcheckerr.i af\sup2\aferrortxt.i ry\app\rydefrescd.i ry\inc\rydestdefi.i ry\inc\ryrepatset.i ry\inc\ryreplnset.i
Other important files	ry\app\rygenomngp.p

For more information on these Managers, see the chapter on using the Progress Dynamics Managers in the *Progress Dynamics Programming Handbook*.

8.2 Temp-tables used by Repository Managers

Because the Repository Managers coordinate data for the entire framework, they make extensive use of temp-tables. They references several temp-tables defined elsewhere in the Progress ADE, including those in the following files:

- adm2\calltables.i
- adm2\ttaction.i
- adm2\tttoolbar.i
- adm2\tttranslate.i

8.2.1 Temp-tables used by Repository Manager

The Repository Manager uses the temp-tables described in [Table 8–3](#) for object retrieval in the runtime environment. The transportClass temp-table is defined like the ttClass temp-table.

Table 8–3: Temp-tables defined in ry\inc\getobject.i

(1 of 2)

Temp-table	Fields (data types)
ttClass	ClassName (Character) ClassObj (Decimal) ClassTableName (Character) ClassBufferHandle (Handle) InheritsFromClasses (Character) SuperProcedures (Character) SuperProcedureModes (Character) SuperHandles (Character) InstanceBufferHandle (Handle) EventTableName (Character) EventBufferHandle (Handle) SetList (Character) GetList (Character) RuntimeList (Character)
transportClass	ClassName (Character) ClassObj (Decimal) ClassTableName (Character) ClassBufferHandle (Handle) InheritsFromClasses (Character) SuperProcedures (Character) SuperProcedureModes (Character) SuperHandles (Character) InstanceBufferHandle (Handle) EventTableName (Character) EventBufferHandle (Handle) SetList (Character) GetList (Character) RuntimeList (Character)
ttEntity	EntityName (Character) EntityTableName (Character) EntityBufferHandle (Handle) LanguageCode (Character)

Table 8–3: Temp-tables defined in ry\inc\getobject.i*(2 of 2)*

Temp-table	Fields (data types)
cacheObject	ObjectName (Character) InstanceId (Decimal) ContainerInstanceId (Decimal) ClassName (Character) AttrOrdinals (Character) AttrValues (Character) Order (Integer) PageNumber (Integer) EventNames (Character) EventActions (Character) ObjectTranslated (Logical) ObjectSecured (Logical)
cachePage	InstanceId (Decimal) PageNumber (Integer) PageLabel (Character) LayoutCode (Character) SecurityToken (Character) TOC (Character) PageReference (Character)
cacheLink	InstanceId (Decimal) SourceInstanceName (Character) TargetInstanceName (Character) LinkName (Character) SourcePage (Integer) TargetPage (Integer)
ttEntityDump	tEntityName (Character) tFieldName (Character) tDataType (Character) tFormat (Character) tLabel (Character) tColLabel (Character) tHelp (Character) tInitial (Character)

The Repository Manager uses the temp-tables described in [Table 8–4](#) to cache attribute information.

Table 8–4: Temp-tables defined in ry\inc\ryrepatset.i

Temp-table	Fields (data types)
ttStoreAttribute	tAttributeParent (Character) tAttributeParentObj (Decimal) tAttributeLabel (Character) tConstantValue (Logical) tCharacterValue (Character) tDecimalValue (Decimal) tIntegerValue (Integer) tDateValue (Date) tRawValue (Raw) tLogicalValue (Logical)
ttStoreUiEvent	tEventParent (Character) tEventParentObj (Decimal) tEventName (Character) tActionType (Character) tActionTarget (Character) tEventAction (Character) tEventParameter (Character) tEventDisabled (Logical) tConstantValue (Logical)

The Repository Manager uses the temp-table described in [Table 8–5](#) to cache information about toolbar bands.

Table 8–5: Temp-tables defined in ry\appl\rymenufunc.i

Temp-table	Fields (data types)
ttBandToExtract	parent_menu_structure_code (Character) menu_structure_type (Character) menu_structure_obj (Decimal) menu_structure_code (Character) extract_sequence (Integer) disabled (Logical) under_development (Logical) menu_item_obj (Decimal) control_spacing (Integer) control_padding (Integer)

Because the Repository Manager coordinates data for the other Managers during a session, it makes use of the other Managers’ temp-tables. [Table 8–6](#) lists these temp-tables and where they are described in this reference manual.

Table 8–6: Other temp-tables used by the Repository Manager

Temp-table	Described in
ttSecurityControl	“Temp-Tables used by Security Manager”
ttTranslate	“Temp-tables used by Localization Manager”
ttUser	“Temp-tables used by General Manager”

8.2.2 Temp-tables used by Repository Design Manager

The Repository Design Manager makes extensive use of temp-tables. It uses the temp-tables described in [Table 8–4](#). It also uses the ttEntityMnemonic temp-table described in the [“Temp-tables used by General Manager”](#) section of [Chapter 4](#), [“General Manager.”](#)

The Repository Design Manager uses the temp-table described in [Table 8–7](#) to cache information about Progress SmartLinks™.

Table 8–7: Temp-tables defined in ry\inc\ryreplnset.i

Temp-table	Fields (data types)
ttSmartLink	tContainerObj (Decimal) tLinkName (Character) tUserLinkName (Character) tSourceObj (Decimal) tTargetObj (Decimal)

The Repository Design Manager uses the temp-table described in [Table 8–8](#) during object generation.

Table 8–8: Temp-tables defined in `ry\app\rygenomngp.p`

(1 of 2)

Temp-table	Fields (data types)
ttDeleteAttribute ¹	tAttributeParent (Character) tAttributeParentObj (Decimal) tAttributeLabel (Character) tConstantValue (Logical) tCharacterValue (Character) tDecimalValue (Decimal) tIntegerValue (Integer) tDateValue (Date) tRawValue (Raw) tLogicalValue (Logical)
ttDataObjectField	tDatabaseName (Character) tTableName (Character) tDumpName (Character) tFieldName (Character) tFieldOrder (Character) tKeepField (Logical) tIsTableObjField (Logical) tFieldUpdatable (Logical)
ttTableColumn	tDatabaseName (Character) tTableName (Character) tDataColumns (Character) tUpdatableColumns (Character) tWhereClause (Character) tOrder (Integer)
ttRelate	cOwnerTable (Character) cDataBaseName (Character) cRelatedTable (Character) cLinkFieldName (Character) cIndexName (Character)

Table 8–8: Temp-tables defined in ry\app\rygenomngp.p (2 of 2)

Temp-table	Fields (data types)
ttTableField	LIKE _field ² tDataBaseName (Character) tTableName (Character) tTableDumpName (Character) tKeyField (Logical) tEntityObjectField (Logical)
ttFrameField	tFieldName (Character) tDBFieldName (Character) tFieldLength (Decimal) tLabelLength (Decimal) tLabelChars (Decimal) tLabel (Character) tTotalFieldWidth (Decimal) tColumn (Decimal) tRow (Decimal) tOrder (Integer) tFieldHandle (Handle) tLabelHandle (Handle) tViewAs (Character) tEnabled (Logical) tDataType (Character) tFormat (Character) tTableName (Character) tTableDumpName (Character) tSortOrder (Character) tHelp (Character) tInitialValue (Character) tSdoOverrides (Character) tKeyField (Logical) tEntityObjectField (Logical)

¹ This temp-table is defined like the ttStoreAttribute temp-table.

² This temp-table starts with a list of the fields for a particular Repository table and adds these extra fields. Its structure depends on the Repository table for which an object is being generated.

The Repository Design Manager APIs use the temp-tables described in [Table 8–9](#) for generating objects.

Table 8–9: Temp-tables defined in `ry\applrydesmgrp.i`

Temp-table	Fields (data types)
ttDeleteAttribute ¹	tAttributeParent (Character) tAttributeParentObj (Decimal) tAttributeLabel (Character) tConstantValue (Logical) tCharacterValue (Character) tDecimalValue (Decimal) tIntegerValue (Integer) tDateValue (Date) tRawValue (Raw) tLogicalValue (Logical)
ttTableField	LIKE _field ² tDataBaseName (Character) tTableName (Character) tTableDumpName (Character) tKeyField (Logical) tEntityObjectField (Logical)
ttXref	tElementType (Character) tSourceData (Character) tTargetData (Character)

¹ This temp-table is defined like the ttStoreAttribute temp-table.

² This temp-table starts with a list of the fields for a particular Repository table and adds these extra fields. Its structure depends on the Repository table for which an object is being generated.

8.3 Repository Manager APIs

This section lists the APIs that you can use with the Repository Manager.

8.3.1 calculateObjectPaths

This procedure is the standard mechanism for finding the paths associated with an object or product module. Use this procedure to calculate relative paths for new objects, existing objects, and associated files for objects, such as a DynSDO's include file.

If an object name is supplied, the procedure looks for its ryc_smartobject.smartobject_obj value. If the smartobject_obj value is known, the object name is not needed. For general path enquiries on a product module, only the product module name is needed.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT pcObjectName AS CHARACTER

The name of the object to be parsed.

INPUT pcObjectObj AS DECIMAL

The smartobject_obj of the object being parsed.

INPUT pcObjectType AS CHARACTER

The gsc_object_type.object_type_code for the object. (optional)

INPUT pcProductModule AS CHARACTER

The name of the product module for the object being parsed. This must be a valid gsc_product_module.product_module_code value. If a valid object name or smartobject_obj is passed in, the product module under which the object is registered is used instead of any passed in value. If only the product module is passed in, the relative path information for the product module is returned.

INPUT pcObjectparameter AS CHARACTER

This field is normally left blank for parsing objects directly. If set to "include", the procedure calculates the file name for an include file associated with the object being parsed. If set to "clientProxy", the procedure calculates the _cl client proxy file name for the object being parsed.

INPUT pcNameSpace AS CHARACTER

Reserved for future use.

OUTPUT pcRootDirectory AS CHARACTER

The calculated root directory for the current session.

OUTPUT pcRelativeDirectory AS CHARACTER

The calculated relative directory based on the product module.

OUTPUT pcSCMRelativeDirectory AS CHARACTER

The calculated SCM relative directory based on the product module. If a SCM tool is being used and SCM checks are in place, the relative path information is retrieved from the currently used SCM tool.

OUTPUT pcFullPathName AS CHARACTER

The calculated full pathname for creating or accessing physical files. The full path does not include the object name. It includes the root directory and either the relative path from the SCM tool if this is valid or the relative directory from the gsc_product_module table.

OUTPUT pcOutputObjectName AS CHARACTER

The validated object name. If the object exists, this returns the repository name of the object, ryc_smartobject.object_filename, in the Repository. If the object does not exist, this parameter contains the pcObjectname value without any relative path information.

OUTPUT pcFileName AS CHARACTER

The calculated physical file name based on the used input parameters. This can be used to create or access the physical file names of the object being parsed or the associated file for the object.

OUTPUT pcError AS CHARACTER

Any errors encountered during processing that need to be returned, such as invalid pcObjectobj values.

Notes: The pcObjectType parameter is not used currently. It is included to allow for future development needs.

Examples: See the getClientCacheDir procedure in ry\app\ryrepmgrp.i.

8.3.2 classHasAttribute

This function checks whether or not a specified attribute or event exists for a class.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT pcClassName AS CHARACTER

INPUT pcAttributeName AS CHARACTER

INPUT plAttributeIsEvent AS LOGICAL

Returns: LOGICAL

Notes: None

Examples: See the rowObjectValidate procedure in ry\obj\rycavlog3p.p.

8.3.3 classIsA

This function checks whether or not a specific class inherits from another specified class.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT pcClassName AS CHARACTER

INPUT pcInheritsFromClass AS CHARACTER

Returns: LOGICAL

Notes: None

Examples: See the generateCalculatedField procedure in ry\app\rygenomngp.p.

8.3.4 clearClientCache

This procedure ensures up-to-date information is retrieved from the database by emptying the client cache temp-tables. This procedure might be called when Repository maintenance programs have run. By using this procedure, you can avoid having to log off and start a new session in order to use the new Repository data settings.

Location: ry\app\ryrepmgrp.i

Parameters: None

Notes: None

Examples: See the cacheObjectOnClient function in ry\app\ryrepmgrp.i.

8.3.5 createClassCache

This procedure caches class attributes and UI events.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT pcClassName AS CHARACTER

A comma-delimited list of class codes (object types), or an asterisk (*) for all.

Notes: If this procedure uses retrieveClassCache procedure, instead of the buildClassCache, to populate the ttClass temp-table, the UI event table is always the first table returned.

Examples: See the getCacheClassBuffer function in ry\app\ryrepmgrp.i.

8.3.6 destroyClassCache

This procedure destroys the temp-tables that make up the class cache, rather than emptying them as clearClientCache does.

Location: ry\app\ryrepmgrp.i

Parameters: None

Notes: This is a separate API because in a runtime environment the class attributes are unlikely to change much.

Examples: See the saveEntitiesToClientCache procedure in ry\app\ryrepmgrp.i.

8.3.7 extractRootFile

This procedure extracts the root filename from a path. It returns both the filename and the filename with an extension, if there is one.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT pcFileName AS CHARACTER

The filename to parse.

OUTPUT pcRootFile AS CHARACTER

The root filename without its extension.

OUTPUT pcRootFileExt AS CHARACTER

The root filename with its extension, if any.

Notes: None

Examples: See the insertObjectMaster procedure in ry\app\rydesmgrp.i.

8.3.8 getCacheClassBuffer

This function returns the buffer handle of the table used to store the cached class buffers.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT pcClassName AS CHARACTER

A class name, a comma-delimited list of class names, or an asterisk (*) for all.

Returns: HANDLE

Notes: If a non-blank and non-null class name is passed in, the ttClass temp-table is repositioned to that record. However, if a list of class names or the wildcard is passed in, the record might not be correctly repositioned.

Examples: See the classHasAttribute function in ry\app\ryrepmgrp.i.

8.3.9 **getCacheLinkBuffer**

This function returns the buffer handle of the table used to cache Repository object links.

Location: ry\app\ryrepmgrp.i

Parameters: None

Returns: HANDLE

Notes: None

Examples: See the fetchUI procedure in ry\app\ryuimsrvrp.p.

8.3.10 **getCacheObjectBuffer**

This function returns the buffer handle of the table used to cache Repository objects. It attempts to find the record specified in pdInstanceID. That value corresponds to the tRecordIdentifier which is unique for each cache_Object record.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT pdInstanceID AS DECIMAL

Returns: HANDLE

Notes: If a null (?) is passed in, the find is ignored.

Examples: See the prepareInstance function in ry\app\ryrepmgrp.i.

8.3.11 **getCachePageBuffer**

This function returns the buffer handle of the table used to cache Repository object pages.

Location: ry\app\ryrepmgrp.i

Parameters: None

Returns: HANDLE

Notes: None

Examples: See the fetchUI procedure in ry\app\ryuimsrvrp.p.

8.3.12 **getClassChildren**

This function returns a comma-delimited list of the class names that are children of the passed-in class name. It retrieves the information from the cache in the ttClass temp-table.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT pcClassName AS CHARACTER

Returns: CHARACTER

Notes: If a class has not been cached, it will not show up in the return list from this function. The getClassChildrenFromDB function performs a similar task, but retrieves the information directly from the Repository.

Examples: See the getOpenObjectFilter function in af\cod2\aftermewizow.w.

8.3.13 **getClassFromInstance**

This function returns the class name of a known object buffer instance.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT pdInstance AS DECIMAL

 The object buffer instance.

Returns: CHARACTER

Notes: None

8.3.14 **getClientCacheDir**

This procedure finds the full absolute path to the client cache directory, based on input product module. The procedure first checks the client_cache_directory session parameter and, if found, returns the session parameter value.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT pcProductModule AS CHARACTER

The product module.

OUTPUT pcFullPathName AS CHARACTER

The full path name to the directory.

Notes: None

Examples: See the createClassCache procedure in ry\app\ryrepmgrp.i.

8.3.15 getCurrentLogicalName

This function returns the name of the object being launched. The prepareInstance function does a call-back to this function to determine the name of the logical object which is being launched.

Location: ry\app\ryrepmgrp.i

Parameters: None

Returns: CHARACTER

Notes: The value of the gcCurrentLogicalName variable is set in the startDataObject procedure.

Examples: See the prepareInstance function in ry\app\ryrepmgrp.i.

8.3.16 getMappedFilename

This function returns the name of the generated file for a logical object name.

The generated file must exist in the 'gen' subdirectory of the directory specified by the client_cache_directory session property if available. If the file cannot be found, the unknown value (?) is returned.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT pcObjectName AS CHARACTER

Returns: CHARACTER

Notes: Only the names of r-code files are returned.

Examples: See the startDataObject procedure in ry\app\ryrepmgrp.i.

8.3.17 getObjectNames

This procedure finds the physical name and logical name of a given object.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT pcObjectName AS CHARACTER

The object name.

INPUT pcRunAttribute AS CHARACTER

The run attribute for the object.

OUTPUT pcPhysicalName AS CHARACTER

The object's physical name.

OUTPUT pcLogicalName AS CHARACTER

The object's logical name.

Notes: None

Examples: See the launchContainer procedure in af\app\afsesmgrp.i.

8.3.18 getObjectSuperProcedure

This procedure returns an objects custom super procedure to a caller.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT pcObjectName AS CHARACTER

The object name.

INPUT pcRunAttribute AS CHARACTER

The run attribute for the object.

OUTPUT pcCustomSuperProc AS CHARACTER

The name of the custom super procedure.

Notes: None

Examples: See the launchContainer procedure in af\app\afsesmgrp.i.

8.3.19 getToolbarBandActions

This procedure returns temp-tables of selected bands and actions to the caller.

Location: ry\app\ryrepmngrp.i

Parameters:

INPUT pcToolbar AS CHARACTER

INPUT pcObjectList AS CHARACTER

INPUT pcBandList AS CHARACTER

A comma-delimited list of band names.

OUTPUT TABLE FOR ttToolbarBand

OUTPUT TABLE FOR ttObjectBand

OUTPUT TABLE FOR ttBand

OUTPUT TABLE FOR ttBandAction

OUTPUT TABLE FOR ttAction

OUTPUT TABLE FOR ttCategory

Notes: These band actions are not cached into a temp-table here because they are cached by the toolbar super procedure, toolbarcustom.p, for the current session.

Examples: See the doToolbar procedure in ry\app\ryuimsrvrp.p.

8.3.20 **IsA**

This function checks whether a class inherits from a particular class, based on the class name.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT pdInstanceId AS DECIMAL

INPUT pcClassName AS CHARACTER

Returns: LOGICAL

Notes: It returns the null value if the object cannot be found in the cache. An object's class cannot be determined if the object is not in the cache.

Examples: See the doViewer procedure in ry\app\ryuimsrvrp.p.

8.3.21 **plipShutdown**

This procedure is a standard part of the Progress Dynamics Manager template. It runs on close of the procedure.

Location: ry\app\ryrepmgrp.i

Parameters: None

Notes: This procedure currently contains no active code.

8.3.22 **resolveResultCodes**

This procedure resolves the result code string. The procedure ensures that the result code string contains valid result codes, including the default result code.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT p1DesignMode AS LOGICAL

INPUT-OUTPUT pcResultCodes AS CHARACTER

Notes: None

Examples: See the ICFCFM_LoginComplete procedure in ry\app\afcusrmgrp.i.

8.3.23 startDataObject

This procedure fetches the specified DataObject from the Repository and starts the object on the client.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT pcDataObject AS CHARACTER

Name of the Data Object

OUTPUT phSDO AS HANDLE

Handle of the started SDO

Notes: If the DataObject is a dynamic SDO, the procedure constructs the necessary attributes.

Examples: See the generateDataFields procedure in ry\app\rydesmgrp.i.

8.3.24 storeAttributeValues

This procedure checks that the passed-in attributes are allowed to be updated and then stores attribute values.

Location: ry\app\ryrepmgrp.i

Parameters:

INPUT phAttributeValueBuffer AS HANDLE

INPUT TABLE-HANDLE phAttributeValueTable

Notes: This procedure does not allow addition of attribute values to object instances that are contained by abstract classes.

Examples: See the registerSdoFields procedure in ry\app\rydesmgrp.i.

8.4 Repository Design Manager APIs

This section lists the APIs that you can use with the Repository Design Manager.

8.4.1 **changeObjectInstance**

This procedure changes the object instance.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcNameContainer AS CHARACTER

The container name. Takes a valid container name or a value of 'ALL' for all containers.

INPUT pcResultCode AS CHARACTER

The customization result code. Takes a valid result code, the name of a Master container, or unknown (?) for all result codes.

INPUT pcNameObjectInstance AS CHARACTER

INPUT pcNameObjectSource AS CHARACTER

The name of the source object to be replaced with the target object. Takes a valid object name.

INPUT pcNameObjectTarget AS CHARACTER

The name of the target object to replace the source object. Takes a valid object name.

INPUT p1DeleteObjectSource AS LOGICAL

Deletes the source object if all instances have been changed and there are no other instance allocations.

INPUT p1RemoveDefaultAttributes AS LOGICAL

Removes default attributes set for the object instances.

INPUT p1RemoveUnusedAttributes AS LOGICAL

Removes unused attributes set for the object instances.

OUTPUT piReplacementCount AS INTEGER

Notes:

- If both the source and target objects are valid objects and exist within the same class hierarchy, the instance and its attribute values are updated accordingly.
- Unless you specify a container name, the procedure works for all containers. If you specify a container name, it works only for the relevant container.
- If the object exists on a container multiple times, all instances are changed unless a specific object instance name is given.
- All the attribute values contained in the attribute value table that are set against the INSTANCE owner are set against this object.

Examples: See the `trgbuReplace` procedure in `ry\prc\ryreplinstvp.p`.

8.4.2 `changeObjectType`

This procedure changes the object type of an object to another object type.

Location: `ry\app\rydesmgrp.i`

Parameters:

INPUT `pcObjectFileName` AS CHARACTER

The object's filename.

INPUT `pcObjectTypeCode` AS CHARACTER

The new object type code.

INPUT `p1RemoveDefaultAttr` AS LOGICAL

If YES, remove attribute values stored against the object that have the same values as the attribute stored against the object's class anywhere in the class hierarchy.

INPUT `p1RemoveNonOTAttr` AS LOGICAL

If YES, remove attribute values stored against the object that are not stored against the object's class anywhere in the class hierarchy.

Notes: None

Examples: See the `changeObjectType` procedure in `ry\app\ryrepmngrp.i`.

8.4.3 classHasAttribute

This function checks whether or not a specified attribute or event exists for a class.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcClassName AS CHARACTER

The class name.

INPUT pcAttributeOrEventName AS CHARACTER

The name of the attribute or event to be checked.

INPUT plAttributeIsEvent AS LOGICAL

Whether or not pcAttributeOrEventName is an event.

Returns: LOGICAL

Notes: There is also a classHasAttribute function in Repository Manager. But that function only checks for runtime attributes. Use this function in tools that need to check design time attributes.

Examples: See the generateCalculatedField procedure in ry\app\rygenomngp.p.

8.4.4 clearDesignCache

This procedure empties the ttClassExtinformation temp-table.

Location: ry\app\rydesmgrp.i

Parameters: None

Notes: None

Examples: See the main block in ry\app\rydesmgrp.i.

8.4.5 copyObjectMaster

This procedure creates a deep copy of an object. This procedure can be used for "Save as . . ." functionality.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcSourceObjectName AS CHARACTER

The original object's name.

INPUT pcSourceResultCode AS CHARACTER

The original object's result code, if any.

INPUT pcTargetObjectName AS CHARACTER

The new object's name.

INPUT pcTargetClass AS CHARACTER

The new object's class.

INPUT pcTargetProductModule AS CHARACTER

The product module in which to save the new object.

INPUT pcTargetRelativePath AS CHARACTER

The relative path for the new object's product module.

OUTPUT pdSmartObjectObj AS DECIMAL

The smartobject_obj value for the new object.

Notes: If a non-default result code is specified, then the procedure ensures that a record exists for the DEFAULT-RESULT-CODE and the specified result code. This ensures that the newly created object is a complete object on its own.

8.4.6 generateCalculatedField

This procedure creates a calculated field.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcCalcFieldName AS CHARACTER

INPUT pcDataType AS CHARACTER

INPUT pcFieldFormat AS CHARACTER

INPUT pcFieldLabel AS CHARACTER

INPUT pcFieldHelp AS CHARACTER

INPUT pcProductModuleCode AS CHARACTER

INPUT pcResultCode AS CHARACTER

INPUT pcObjectTypeCode AS CHARACTER

Notes: None

Examples: See the main block in ry\app\rydesgecfp.p.

8.4.7 generateClassCache

This procedure outputs the class cache to the disk.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcClassList AS CHARACTER

A comma delimited list of classes to output.

OUTPUT pcStatus AS CHARACTER

The status of the class cache.

Notes: None

Examples: See the generateCache procedure in ry\obj\ryclcrunv.w.

8.4.8 generateDataFields

This procedure creates DataField records for a table. This procedure creates a SmartDataField for each field in the table.

You can pass in a list of tables for which to generate fields. This means that you can exclude certain tables, instead of having to specify all tables. For example, if all tables except one are to be generated, a string like “!SomeTable,*” can be passed into the procedure.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcDataBaseName AS CHARACTER

INPUT pcTableName AS CHARACTER

A comma-separated list of tables, in CAN-DO () format, that are to have fields generated.

INPUT pcProductModuleCode AS CHARACTER

INPUT pcResultCode AS CHARACTER

INPUT plGenerateFromDataObject AS LOGICAL

INPUT pcDataObjectFieldList AS CHARACTER

INPUT pcSdoObjectName AS CHARACTER

INPUT pcObjectTypeCode AS CHARACTER

INPUT pcOverrideAttributes AS CHARACTER

INPUT pcFieldNames AS CHARACTER

Notes: None

Examples: See the main block in af\app\afgengenob.i.

8.4.9 generateDataLogicObject

This procedure generates a DataLogic object procedure.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcDatabaseName AS CHARACTER

INPUT pcTableName AS CHARACTER

INPUT pcDumpName AS CHARACTER

INPUT pcDataObjectName AS CHARACTER

INPUT pcProductModule AS CHARACTER

INPUT pcResultCode AS CHARACTER

INPUT plSuppressValidation AS LOGICAL

INPUT pcLogicProcedureName AS CHARACTER

INPUT pcLogicObjectType AS CHARACTER

INPUT pcLogicProcedureTemplate AS CHARACTER

INPUT pcDataObjectRelativePath AS CHARACTER

INPUT pcDataLogicRelativePath AS CHARACTER

INPUT pcRootFolder AS CHARACTER

INPUT pcFolderIndicator AS CHARACTER

INPUT plCreateMissingFolder AS LOGICAL

Notes: None

Examples: See the main block in af\app\afgengenob.i.

8.4.10 generateDataObject

This procedure generates SDOs and other data objects.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcDatabaseName AS CHARACTER

INPUT pcTableName AS CHARACTER

INPUT pcDumpName AS CHARACTER

INPUT pcDataObjectName AS CHARACTER

INPUT pcObjectTypeCode AS CHARACTER

INPUT pcProductModule AS CHARACTER

INPUT pcResultCode AS CHARACTER

INPUT plCreateSDODataFields AS LOGICAL

INPUT plSdoDeleteInstances AS LOGICAL

INPUT plSuppressValidation AS LOGICAL

INPUT plFollowJoins AS LOGICAL

INPUT piFollowDepth AS INTEGER

INPUT pcFieldSequence AS CHARACTER

INPUT pcLogicProcedureName AS CHARACTER

INPUT pcDataObjectRelativePath AS CHARACTER

INPUT pcDataLogicRelativePath AS CHARACTER

INPUT pcRootFolder AS CHARACTER

INPUT plCreateMissingFolder AS LOGICAL

INPUT pcAppServerPartition AS CHARACTER

Notes: The main code for this procedure is in the include file,
ry\app\rydesmngdo.i.

Examples: See the main block in af\app\afgengenob.i.

8.4.11 generateDynamicBrowse

This procedure generates a dynamic browse based on the information obtained from the object generator.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcObjectTypeCode AS CHARACTER

INPUT pcObjectName AS CHARACTER

INPUT pcObjectDescription AS CHARACTER

INPUT pcProductModuleCode AS CHARACTER

INPUT pcResultCode AS CHARACTER

INPUT pcSdoObjectName AS CHARACTER

INPUT plDeleteExistingInstances AS LOGICAL

INPUT pcDisplayedDatabases AS CHARACTER

INPUT pcEnabledDatabases AS CHARACTER

INPUT pcDisplayedTables AS CHARACTER

INPUT pcEnabledTables AS CHARACTER

INPUT pcDisplayedFields AS CHARACTER

INPUT pcEnabledFields AS CHARACTER

INPUT piMaxFieldsPerColumn AS INTEGER

INPUT pcDataObjectFieldSequence AS CHARACTER

INPUT pcDataObjectFieldList AS CHARACTER

OUTPUT pdVisualObjectObj AS DECIMAL

Notes: None

Examples: See the main block in ry\app\rydesgedbp.p.

8.4.12 generateDynamicSDF

This procedure generates a dynamic Progress SmartDataField™ (SDF) based on the information obtained from the SDF Maintenance Tool.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcObjectName AS CHARACTER

INPUT pcObjectDescription AS CHARACTER

INPUT pcProductModuleCode AS CHARACTER

INPUT pcResultCode AS CHARACTER

INPUT plDeleteExistingInstances AS LOGICAL

INPUT pcSDFTYPE AS CHARACTER

Acceptable values are DynCombo or DynLookup.

INPUT pcSuperProcedure AS CHARACTER

The name of the super procedure.

INPUT pcAttributeLabels AS CHARACTER

A CHR(1) delimited list of attribute names.

INPUT pcAttributeValues AS CHARACTER

A CHR(1) delimited list of attribute values corresponding to attributes names from pcAttributeLabels.

INPUT pcAttributeDataType AS CHARACTER

A CHR(1) delimited list of attribute data type corresponding to attributes names from pcAttributeLabels, as specified in af\app\afdatatypi.i.

OUTPUT pdSDFObjectObj AS DECIMAL

Notes: None

Examples: See the saveComboDetails procedure in ry\obj\rydynsdfmv.w.

8.4.13 generateDynamicViewer

This procedure generates a dynamic viewer based on the information obtained from the object generator.

Location: ry\app\rydesmngpr.i

Parameters:

INPUT pcObjectTypeCode AS CHARACTER

INPUT pcObjectName AS CHARACTER

INPUT pcObjectDescription AS CHARACTER

INPUT pcProductModuleCode AS CHARACTER

INPUT pcResultCode AS CHARACTER

INPUT pcSdoObjectName AS CHARACTER

INPUT plDeleteExistingInstances AS LOGICAL

INPUT pcDisplayedDatabases AS CHARACTER

INPUT pcEnabledDatabases AS CHARACTER

INPUT pcDisplayedTables AS CHARACTER

INPUT pcEnabledTables AS CHARACTER

INPUT pcDisplayedFields AS CHARACTER

INPUT pcEnabledFields AS CHARACTER

INPUT piMaxFieldsPerColumn AS INTEGER

INPUT pcDataObjectFieldSequence AS CHARACTER

INPUT pcDataObjectFieldList AS CHARACTER

OUTPUT pdVisualObjectObj AS DECIMAL

Notes: The main code for this procedure is in the include file,
ry\app\rydesmngdv.i.

Examples: See the main block in ry\app\rydesgedvp.p.

8.4.14 generateEntityInstances

This procedure associates DataFields and other master objects with an entity.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcEntityObjectName AS CHARACTER

The entity object name, which is the same as the entity mnemonic.

INPUT pcFieldList AS CHARACTER

A CHR(3) delimited list of fields in the order they should be added to the entity.

INPUT plDeleteExistingInstances AS LOGICAL

Whether to remove any existing instances before adding those specified here.

Notes:

- Entities cannot be customized, so they have no result code.
- Because the objects to be associated with the fields are not bound to the schema and passing in a table name limits this API, the field list is used as passed in without performing any calculations. The caller should determine which master objects (dataField or other) to make instances of the entity.

Examples: See the main block in ry\app\rydesgeeip.p.

8.4.15 generateEntityObject

This procedure generates an object for an entity, using a table in the database.

Location: ry\app\rydesmngpr.i

Parameters:

INPUT pcTableName AS CHARACTER

A list of databases and tables from which to generate objects. The list is in the following format:

*<database name> CHR(3) <first tablename>,<next tablename>, . . . <last tablename>
CHR(3)
<next database name> . . .*

INPUT pcEntityType AS CHARACTER

The class name of the Entity object. It must be descended from the Entity class.

INPUT pcEntityProductModule AS CHARACTER

The product module into which to create the Entity object.

INPUT plAutoProPerform AS LOGICAL

INPUT piPrefixLength AS INTEGER

INPUT pcSeparator AS CHARACTER

Valid values are Upper, blank, or a printable character.

INPUT pcAuditingEnabled AS CHARACTER

Valid values are (Y)es, (N)o, or (I)gnore

INPUT pcDescFieldQualifiers AS CHARACTER

Determines the criteria used to search for the entity description field. If blank, uses the defaults.

INPUT pcKeyFieldQualifiers AS CHARACTER

Determines the criteria used to search for the entity key field. If blank, uses the defaults.

INPUT pcObjFieldQualifiers AS CHARACTER

Determines the criteria used to search for the entity object field. If blank, uses the defaults.

INPUT plDeployData AS LOGICAL

INPUT plVersionData AS LOGICAL

INPUT plReuseDeletedKeys AS LOGICAL

INPUT plAssociateDataFields AS LOGICAL

Whether to automatically associate DataFields with the entity.

Notes: If plAssociateDataFields is set to YES, then all existing datafield instances are first removed.

8.4.16 generateSBODataLogicObject

This procedure generates the DataLogicProcedure (DLP) for the SBO and its client proxy, compiles them, and registers it in the Repository.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcDatabaseName AS CHARACTER

The database name. This parameter is not used.

INPUT pcTableList AS CHARACTER

A comma-delimited list of supported tables.

INPUT pcDumpName AS CHARACTER

This parameter is not used.

INPUT pcDataObjectName AS CHARACTER

The name of SBO object.

INPUT pcProductModule AS CHARACTER

The product module of DLP.

INPUT pcResultCode AS CHARACTER

The result code. If blank, uses the default result code.

INPUT pcLogicProcedureName AS CHARACTER

The name of new logic procedure to be created.

INPUT pcLogicObjectType AS CHARACTER

The logic type of new procedure, such as DLCProc.

INPUT pcLogicProcedureTemplate AS CHARACTER

The template file on which to base DLP.

INPUT pcDataLogicRelativePath AS CHARACTER

The relative path of object to be saved.

INPUT pcRootFolder AS CHARACTER

The root directory, for example, "C:\workarea".

INPUT pcIncludeFileList AS CHARACTER

A comma-delimited list of include files of the SBO including relative path. This must match the list in pcTableList.

INPUT p1CreateMissingFolder AS LOGICAL

If YES, create new directory if relative directory specified does not exist.

Notes: None

Examples: See the main block in ry\app\rydesgsd1p.p.

8.4.17 generateSDOInstances

This procedure associates datafield instances with a given SDO object.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcSdoObjectName AS CHARACTER

INPUT pcResultCode AS CHARACTER

INPUT p1DeleteExistingInstances AS LOGICAL

INPUT pcTableList AS CHARACTER

Notes: Only the Enabled attribute is set here. Because the SDO .i fields are created with the LIKE option, all other attributes are inherited from the schema.

Examples: See the createNewSDO procedure in af\cod2\fullobjcw.w.

8.4.18 generateVisualObject

This procedure creates a dynamic browse, based on a table, SDO, or table and SDO. This procedure is the starting point to create both dynamic browses and viewers from the Object Generator.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcObjectType AS CHARACTER
INPUT pcObjectName AS CHARACTER
INPUT pcProductModuleCode AS CHARACTER
INPUT pcResultCode AS CHARACTER
INPUT pcSdoObjectName AS CHARACTER
INPUT pcTableName AS CHARACTER
INPUT pcDataBaseName AS CHARACTER
INPUT piMaxObjectFields AS INTEGER
INPUT piMaxFieldsPerColumn AS INTEGER
INPUT plGenerateFromDataObject AS LOGICAL
INPUT pcDataObjectFieldList AS CHARACTER
INPUT plDeleteExistingInstances AS LOGICAL
INPUT pcDataObjectFieldSequence AS CHARACTER
INPUT plUseSDOFieldOrder AS LOGICAL
OUTPUT pdVisualObjectObj AS DECIMAL

Notes: None

Examples: See the main block in af\app\afgengenob.i.

8.4.19 **getBufferDbName**

This function returns the database name for a given table.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcTableName AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the generateVisualObject procedure in ry\app\rydesmgrp.i.

8.4.20 **getProductModuleList**

This function returns a list-item-pairs list of product module information for use in product module combo boxes. The default delimiter is CHR(3).

For example, the following code:

```
getProductModuleList(INPUT 'product_module_code',  
    INPUT 'product_module_code,product_module_description,relative_path'  
    INPUT "&1 // &2 (&3) "  
    INPUT CHR(3) ).
```

Returns the following result:

```
"af-aaa // ICF Root Directory (af/aaa)|af-aaa..."
```

Location: ry\app\rydesmgrp.i

Parameters:

pcValueField AS CHARACTER

The field from the product_module table used as the value field in the list-item-pairs

pcDescFields AS CHARACTER

The comma delimited list of product module fields to display in the label portion of the list-item-pairs. The maximum number of fields is three.

pcDescFormat AS CHARACTER

The base string containing substitution parameters of the form &n used to substitute the description fields.

pcDelimiter AS CHARACTER

The delimiter used to build the list-item-pairs. The default is CHR(3).

Returns: CHARACTER

Notes:

- The Query String Filter Set for the session is used to show or not show Repository modules.
- This function replaces deprecated productModuleList API.

Examples: See the initializeData procedure in af\obj2\gscemimptv.w.

8.4.21 getSchemaQueryHandle

This function creates a query for the metaschema.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcDatabaseName AS CHARACTER

INPUT pcTableNames AS CHARACTER

OUTPUT pcWidgetPoolName AS CHARACTER

Returns: HANDLE

Notes: None

Examples: See the buildSchemaFieldTable procedure in ry\app\rydesmgrp.i.

8.4.22 **getWidgetSizeFromFormat**

This function returns the height and width of a FILL-IN widget, based on the format.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcFormatMask AS CHARACTER

INPUT pcUnit AS CHARACTER

The allowed values are CHARACTER or PIXEL. CHARACTER is the default.

OUTPUT pdHeight AS DECIMAL

Returns: DECIMAL

Notes: None

Examples: See the registerSdoFields procedure in ry\app\rydesmgrp.i.

8.4.23 **insertObjectInstance**

This procedure adds or updates an object instance on a container. If the object already exists on the container, the instance and its attribute values are updated accordingly. Otherwise, the instance is created.

Because an object can exist multiple times on a container, this procedure determines whether the object exists based on the object instance name. If the “force new create” flag is set, a new object instance is created even if one already exists.

If an attribute value is contained in the attribute value table and is set against the instance owner, the attribute value is set against the object.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pdContainerObjectObj AS DECIMAL

INPUT pcObjectName AS CHARACTER

INPUT pcResultCode AS CHARACTER

INPUT pcInstanceName AS CHARACTER

INPUT pcInstanceDescription AS CHARACTER

INPUT pcLayoutPosition AS CHARACTER

```
INPUT piPageNumber AS INTEGER

INPUT piPageSequence AS INTEGER

INPUT plForceCreateNew AS LOGICAL

INPUT phAttributeValueBuffer AS HANDLE

INPUT TABLE-HANDLE phAttributeValueTable.

OUTPUT pdSmartObjectObj AS DECIMAL

OUTPUT pdObjectInstanceObj AS DECIMAL
```

Notes: None

Examples: See the registerSdoFields procedure in ry\app\rydesmgrp.i.

8.4.24 insertObjectLinks

This procedure adds or updates an object's links.

Location: ry\app\rydesmgrp.i

Parameters:

```
INPUT dContainerObjObjectObj AS DECIMAL
```

This parameter has been deprecated. It remains in the API signature for compatibility reasons.

```
INPUT phSmartLinkBuffer AS HANDLE
```

```
INPUT TABLE-HANDLE phSmartLinkTable
```

Notes: None

Examples: See the main block in ry\app\rydescpmoi.i.

8.4.25 insertObjectMaster

This procedure creates or updates records on the Repository's ryc_smartobject table for the passed-in dynamic or static object. If an attribute value is contained in the attribute value table and is set against the master owner, the attribute value is set against the object.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcObjectName AS CHARACTER

The name of the object to be stored.

INPUT pcResultCode AS CHARACTER

The result code for which to store this object. A blank result code defaults to the DEFAULT-RESULT-CODE. This is an optional parameter.

INPUT pcProductModuleCode AS CHARACTER

The product module in which to store the object.

INPUT pcObjectTypeCode AS CHARACTER

The object type (class) of this object.

INPUT pcObjectDescription AS CHARACTER

A description of the object.

INPUT pcObjectPath AS CHARACTER

The relative path of this object. If this is blank, then the product module determines the relative path of the object. This is an optional parameter.

INPUT pcSdoObjectName AS CHARACTER

The name of an SDO associated with this object. This is an optional parameter.

INPUT pcSuperProcedureName AS CHARACTER

The name of a super procedure associated with this object. This is an optional parameter. Note that passing in a blank or unknown (?) value does not remove any existing superprocedure.

INPUT plIsTemplate AS LOGICAL

Is this object a template?

INPUT plIsStatic AS LOGICAL

Is this a static object?

INPUT pcPhysicalObjectName AS CHARACTER

The name of the physical object associated with this object. Except for static objects, a physical object must be supplied. If none is passed in, a default is used. This is an optional parameter.

INPUT plRunPersistent AS LOGICAL

Whether or not the object must run persistently.

INPUT pcTooltipText AS CHARACTER

The tooltip text for the object. If this is blank, the object description is used.

INPUT pcRequiredDBList AS CHARACTER

The databases required to run this object.

INPUT pcLayoutCode AS CHARACTER

The layout code for this object. This is an optional parameter.

INPUT phAttributeValueBuffer AS HANDLE

Pointers to the attribute value temp-table.

INPUT TABLE-HANDLE phAttributeValueTab1

OUTPUT pdSmartObjectObj AS DECIMAL

The smartobject_obj of the updated object.

Notes:

- If the object type of the object has the “layout supported” flag set to YES, the layout defaults to RELATIVE if none is specified.
- This procedure is not intended to remove superprocedures from objects. If you pass in a blank or unknown (?) value to pcSuperProcedureName, the procedure does nothing. To remove a superprocedure, use the removeAttributeValues procedure.

Examples: See the generateDataFields procedure in ry\app\rydesmgrp.i.

8.4.26 insertObjectPage

This procedure creates and updates an object page.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcContainerObjectName AS CHARACTER

The object name of the container to which the page belongs.

INPUT pcContainerResultCode AS CHARACTER

The container's result code, if any.

INPUT pcPageLabel AS CHARACTER

The page label.

INPUT pcSecurityToken AS CHARACTER

The page's security token.

INPUT pcPageReference AS CHARACTER

The page's reference

INPUT piPageSequence AS INTEGER

The page's sequence

INPUT pcLayoutCode AS CHARACTER

The page layout code.

INPUT pcEnableOn AS CHARACTER

A CAN-DO style list which needs to be able to resolve VIEW,MODIFY and CREATE.

OUTPUT pdPageObj AS DECIMAL

The page_obj value for the new page.

Notes: None

Examples: See the main block in ry\app\rydescpmoi.i.

8.4.27 insertUiEvents

This procedure stores UI Events.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT phUiEventBuffer AS HANDLE

INPUT TABLE-HANDLE phUiEventTable

Notes:

- The buffer used in this procedure are based on the ttStoreUiEvent temp-table defined in ry\inc\ryrepatset.i.
- This procedure does not allow the addition of UI events for object instances that are contained by abstract classes.

Examples: See the main block in ry\app\rydescpmoi.i.

8.4.28 ObjectExists

This function checks if a given object name exists in the Repository.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcObjectName AS CHARACTER

Returns: LOGICAL

Notes: None

Examples: See the main block in ry\inc\rygenogtji.i.

8.4.29 plipShutdown

This procedure is a standard part of the manager template. It runs on close of the procedure.

Location: ry\app\rydesmgrp.i

Parameters: None

Notes: This procedure currently contains no active code.

8.4.30 prepareObjectName

This function enforces naming standards when creating new dynamic objects and writing them to the Repository. All saves pass through this function.

You can create a customized version of this function using the prescribed method of customizing managers.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcObjectName AS CHARACTER

The suggested base name of the object, required if pcAction is SAVE.

INPUT pcResultCode AS CHARACTER

The result code for a customized object.

INPUT pcObjectString AS CHARACTER

An optional string that can pass additional info. Reserved for future use.

INPUT pcAction AS CHARACTER

The type of action requested. SAVE means the object name is being saved to the Repository. DEFAULT means that a default object name is requested.

INPUT pcObjectType AS CHARACTER

The object type being saved.

INPUT pcEntityName AS CHARACTER

The object's entity mnemonic, required if pcAction is DEFAULT.

INPUT pcProductModule AS CHARACTER

The product module for the object, required if pcAction is SAVE.

OUTPUT pcNewObjectName AS CHARACTER

The new object name. This field must be unique, non-blank, and not null.

OUTPUT pcNewObjectExt AS CHARACTER

The object extension if required.

Returns: CHARACTER

Notes: The InsertObjectMaster procedure calls this function using pcAction='Save'. The ObjectGenerator calls this function using pcAction = 'Default'. The AppBuilder Save dialog calls this function using pcAction = 'Default'.

Examples: See the insertObjectMaster procedure in ry\app\rydesmgrp.i.

8.4.31 removeAttributeValues

This procedure removes attribute values.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT phAttributeValueBuffer AS HANDLE

INPUT TABLE-HANDLE phAttributeValueTable

Notes:

- The buffer used in this procedure are based on the ttStoreAttribute temp-table defined in ry\inc\ryrepatset.i.
- This procedure should be used to remove an object's superprocedure, rather than the insertObjectMaster procedure.

Examples: See the propertyChangedAttribute procedure in af\cod2\afmenuaintw.w.

8.4.32 removeObject

This procedure removes an object from the Repository. The ryc_smartobject delete trigger deletes the following records associated with the object:

- attribute values
- menu items
- valid object partitions
- toolbar menu structures
- object menu structures
- smartlinks
- comments
- user allocations
- multi-media

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcObjectName AS CHARACTER

INPUT pcResultCode AS CHARACTER

Notes: When an object instance is deleted, the attribute values and UI events for that object instance are deleted by the delete triggers.

Examples: See the deleteObject procedure in ry\obj\bopendialog.w.

8.4.33 removeObjectInstance

This procedure removes object instances from a container.

The following rules are used to decide which object instances to delete. In all cases outlined below, the passed-in result code determines which object instances are removed. If the ALL-RESULT-CODE value is passed in, all object instances are removed:

1. If the pcInstanceName is set and the pcInstanceObjectName is not set, then all object instance records which have the specified instance name are removed.
2. If the pcInstanceName is not set and the pcInstanceObjectName is set, then all instances which have the given object name are removed.
3. If neither the pcInstanceName or pcInstanceObjectName are set, all object instances for the container are removed.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcContainerObjectName AS CHARACTER

INPUT pcContainerResultCode AS CHARACTER

INPUT pcInstanceObjectName AS CHARACTER

INPUT pcInstanceName AS CHARACTER

INPUT pcInstanceResultCode AS CHARACTER

Notes: The ALL-RESULT-CODE cannot be used for the container object.

Examples: See the generateDynamicBrowse procedure in ry\app\rydesmgrp.i.

8.4.34 removeObjectPage

This procedure removes a page from an object. You can specify the page to remove with either the page reference or the page sequence. The page reference takes precedence if both are supplied.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcContainerObjectName AS CHARACTER

The object name of the container in which the page exists.

INPUT pcContainerResultCode AS CHARACTER

The container result code, if any.

INPUT pcPageReference AS CHARACTER

The page reference. An asterisk, *, indicates all pages should be removed.

INPUT piPageSequence AS INTEGER

The page sequence.

INPUT p1RemoveObjectInstances AS LOGICAL

If YES, remove the object instance record. If NO, only remove the record linking the object instance to a particular page.

Notes: None

8.4.35 **removePageInstance**

This procedure removes a page object record.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT pcContainerObjectName AS CHARACTER

 The object name of the container in which the page exists.

INPUT pcContainerResultCode AS CHARACTER

 The container result code, if any.

INPUT pcPageReference AS CHARACTER

 The page reference.

INPUT pcInstanceName AS CHARACTER

 The instance name.

INPUT plDeleteObjectInstance AS LOGICAL

 If YES, also remove the ryc_object_instance record.

Notes: The procedure assumes a one-to-one relationship is assumed between then ryc_object_instance and ryc_page_object records.

Examples: See the removeObjectPage procedure in ry\app\rydesmgrp.i.

8.4.36 removeUIEvents

This procedure removes UI Events.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT phUiEventBuffer AS HANDLE

INPUT TABLE-HANDLE phUiEventTable

Notes: The buffer used in this procedure are based on the ttStoreUiEvent temp-table defined in ry\inc\ryrepatset.i.

Examples: See the writeFieldLevelObjects procedure in ry\prc\rygendynp.p.

8.4.37 setQualifiedTableName

This function sets the flag that indicates whether or not table names should be qualified with the logical database name.

Location: ry\app\rydesmgrp.i

Parameters:

INPUT p1SuppressDbName AS LOGICAL

If YES, do not qualify the table names.

Returns: LOGICAL

Notes: None

Security Manager

The Security Manager handles retrieving and applying security information from the Repository. This chapter covers the following subjects:

- [Overview](#)
- [Temp-Tables used by Security Manager](#)
- [Security Manager APIs](#)

9.1 Overview

The Security Manager aids in the definition of security allocations and in applying those allocations at run time. Security allocations are grants or restrictions of privileges for specific users. Security allocations can be defined on:

- Actions, which restrict access to UI controls such as toolbar buttons and tabs in a tab folder.
- Restricted tables and fields in a database, including calculated fields.
- Restricted ranges of data values in a particular field.
- Restricted database records filtered in any way.
- Restricted containers, menu structures, menu items, and login companies.

In a distributed environment, the Security Manager is started on both the client and server sides. There is a wrapper program for each side that either sets the global variable, `server-side`, to YES for the server-side version or to NO for the client-side version. Both programs then include the main code of the manager from the `af\app\afsecmgrp.i` include file. The differences in how the manager operates on each side are triggered by checking the value of the `server-side` variable.

The Security Manager's handle is stored in the global shared variable, `gshSecurityManager`.

[Table 9-1](#) shows the files that contain the Security Manager's code.

Table 9-1: Security Manager files

Type	Path
Client-side wrapper	af\sup2\afsecc1ntp.p
Server-side wrapper	af\app\afsecsrvrp.p
Main code	af\app\afsecmgrp.i
Included files	af\app\afsecgtsecctrlp.i af\app\afsecttdef.i af\app\afsecupdusrallocp.i af\app\afstsecurityctrl.i af\app\usrSecChck.i af\sup2\aferrortxt.i af\sup2\afglobals.i

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in the *Progress Dynamics Programming Handbook*. For more information about setting up security, see the chapters on security in the *Progress Dynamics Administration Guide* and the *Progress Dynamics Developer's Guide*.

9.2 Temp-Tables used by Security Manager

The Security Manager uses several temp-tables to control the security allocations. [Table 9–2](#) describes the temp-tables used to cache security information.

Table 9–2: Temp-tables defined in a\app\lafsecmgrp.i

(1 of 2)

Temp-table	Fields (data types)
ttUserSecurityCheck	dUserObj (Decimal) dOrganisationObj (Decimal) cOwningEntityMnemonic (Character) dOwningObj (Decimal) lSecurityCleared (Logical) CSecurityValue1 (Character) CSecurityValue2 (Character)
ttFieldSecurityCheck	cObjectName (Character) cAttributeCode (Character) cSecurityOptions (Character)
ttTokenSecurityCheck	cObjectName (Character) cAttributeCode (Character) cSecurityOptions (Character)
ttTableSecurityCheck	cOwningEntityMnemonic (Character) cEntityFieldName (Character) cValidValues (Character)
ttRangeSecurityCheck	cRangeCode (Character) cObjectName (Character) cAttributeCode (Character) cRangeFrom (Character) cRangeTo (Character)
ttObjectSecurityCheck	cObjectName (Character) dObjectObj (Decimal) lRestricted (Logical)

Table 9–2: Temp-tables defined in a\app\afsecmgrp.i*(2 of 2)*

Temp-table	Fields (data types)
ttMenuSecurity	cName (Character) cMenuType (Character) cSecurityOptions (Character)
ttUser	userObj (Decimal)
ttGlobalSecurityStructure	product_module_obj (Decimal) owning_entity_mnemonic (Character) owning_obj (Decimal) security_structure_obj (Decimal) security_object_name (Character) restricted (Logical) user_allocation_value1 (Character) user_allocation_value2 (Character)
ttGlobalSecurityAllocation	login_organisation_obj (Decimal) owning_entity_mnemonic (Character) owning_obj (Decimal) user_allocation_value1 (Character) user_allocation_value2 (Character)

The Security Manager uses the temp-table described in [Table 9–3](#) to cache security setting information.

Table 9–3: Temp-tables defined in a\app\afsecttdef.i

Temp-table	Fields (data types)
ttUpdatedAllocations	lDeleteAll (Logical) lDelete (Logical) lUpdateValue1AndValue2 (Logical) owning_entity_mnemonic (Character) owning_obj (Decimal) user_allocation_value1 (Character) user_allocation_value2 (Character)

The Security Manager uses the temp-table described in [Table 9–4](#), a copy of the Repository `gsc_security_control` table, to cache security setting information.

Table 9–4: Temp-tables defined in `af\app\lfttsecurityctrl.i`

Temp-table	Fields (data types)
ttSecurityControl	security_control_obj (Decimal) password_max_retries (Integer) password_history_life_time (Integer) full_access_by_default (Logical) security_enabled (Logical) help_writer_enabled (Logical) build_top_menus_only (Logical) default_help_filename (Character) error_log_filename (Character) translation_enabled (Logical) login_filename (Character) multi_user_check (Logical) program_access_check (Logical) minimise_siblings (Logical) enable_window_positioning (Logical) force_unique_password (Logical) company_logo_filename (Character) system_icon_filename (Character) small_icon_filename (Character) product_logo_filename (Character) scm_checks_on (Logical) scm_tool_obj (Decimal) user_context_expiry_period (Decimal)

9.3 Security Manager APIs

This section lists the APIs that you can use with the Security Manager.

9.3.1 areFieldsCached

This function indicates if security fields for a specific object have been cached.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pcObjectName AS CHARACTER

INPUT pcRunAttribute AS CHARACTER

Returns: LOGICAL

Notes: None

Examples: See the containerCacheUpfront procedure in af\app\afsesmgrp.i.

9.3.2 areTokensCached

This function indicates if security tokens for a specific object have been cached.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pcObjectName AS CHARACTER

INPUT pcRunAttribute AS CHARACTER

Returns: LOGICAL

Notes: None

Examples: See the containerCacheUpfront procedure in af\app\afsesmgrp.i.

9.3.3 authenticateUser

This procedure authenticates users. This procedure verifies that a user exists in the database and that the password provided is legitimate. It does not establish any permissions.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pcUserName AS CHARACTER

The user name from the login.

INPUT pcPassword AS CHARACTER

The password from the login.

OUTPUT pcError AS CHARACTER

The error message if the login fails.

Notes: None

Examples: See the main block in icf\as_connect.p.

9.3.4 cacheGlobalSecurityAllocations

This procedure populates the ttGlobalSecurityAllocation temp-table with data from the Repository's gsm_user_allocation table.

Location: af\app\afsecmgrp.i

Parameters: None

Notes: None

Examples: See the clearClientCache procedure in af\app\afsecmgrp.i.

9.3.5 cacheGlobalSecurityStructures

This procedure caches all global security structures at session startup. The cached data can then be used instead of repeatedly reading these records from the Repository.

Location: af\app\afsecmgrp.i

Parameters: None

Notes: None

Examples: See the clearClientCache procedure in af\app\afsecmgrp.i.

9.3.6 changePassword

This procedure changes a user's password, making all the relevant checks to password history.

This procedure first checks that the passed-in user is valid and not disabled (either on the user record or the user category record). The procedure then validates that the old password is correct, similar to the checkUser procedure, and returns an error if it is not valid.

Providing the old password is OK, the new password is then validated according to the rules set up on the system or user record. It checks password minimum length and the password history if enabled. If everything is correct, the new password is saved for the user and the appropriate user details updated. If the password was expired, the expiration details are reset.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pdUserObj AS DECIMAL

The user object number if known.

INPUT pcLoginName AS CHARACTER

The user login name if known.

INPUT pcOldPassword AS CHARACTER

The old password (encoded).

INPUT pcNewPassword AS CHARACTER

The new password (encoded).

INPUT plExpired AS LOGICAL

The password expired flag.

INPUT piLength AS INTEGER

The number of password characters entered in new password.

OUTPUT pcError AS CHARACTER

Any failure reason (standard Dynamics-formatted error).

Notes: None

Examples: See the main block in af\cod2\aftemcpasw.w.

9.3.7 checkUser

This procedure authenticates the passed-in user or company identities.

Because you always want to use the latest information to authenticate the user who is trying to log in, this procedure does not cache. The following checks are made on the user details entered:

1. Checks if the user is valid and, if it is not, returns an error stating that an invalid login name was specified.
2. Checks if the account has been disabled and, if it has, returns an error.
3. Checks if the category of user has been disabled and, if it has, returns an error.
4. If a password was entered, checks that the password is valid for the user. If the password is not valid, then the fail count is updated on the user record. If this exceeds the maximum retries, the account is additionally disabled and the retries reset back to 0.
5. If a valid password is entered, the fail count is reset, and the login details updated on the user.
6. If the user password has expired, then this fact is returned to the login window. The login window should prompt for a new password before proceeding with the login. If this fails, the login is aborted.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pcLoginName AS CHARACTER

The user login name.

INPUT pcPassword AS CHARACTER

The encoded user password.

INPUT pdCompanyObj AS DECIMAL

The login company obj specified.

INPUT pdLanguageObj AS DECIMAL

The language obj specified.

OUTPUT pdUserObj AS DECIMAL

The login user object number.

OUTPUT pcUserName AS CHARACTER

The user full name.

OUTPUT pcUserEmail AS CHARACTER

The user e-mail.

OUTPUT pcOrganisationCode AS CHARACTER

The organization code.

OUTPUT pcOrganisationName AS CHARACTER

The organization name.

OUTPUT pcOrganisationShort AS CHARACTER

The organization short name.

OUTPUT pcLanguageName AS CHARACTER

The language name.

OUTPUT pcError AS CHARACTER

Any failure reason (standard Progress Dynamics-formatted error).

Notes: None

Examples: See the authenticateUser function in ry\app\ryreqsrvrp.p.

9.3.8 clearClientCache

This procedure ensures that up-to-date information is retrieved from the database by emptying the client cache temp-tables. This procedure might be called when security maintenance programs have run. By using this procedure, you can avoid having to log off and start a new session in order to use the new security settings.

Location: af\app\afsecmgrp.i

Parameters: None

Notes: None

Examples: See the relogin procedure in af\app\afsesmgrp.i.

9.3.9 createGroupAllocation

This procedure adds a group allocation for a user.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pdUserObj AS DECIMAL

The user's unique key field.

INPUT pdGroupObj AS DECIMAL

The group's unique key field.

Notes: None

Examples: See the createGroupFromUser procedure in af\app\afsecmgrp.i.

9.3.10 createGroupFromUser

This procedure creates a security group based on a specific user's allocations.

If the user is a profile user, only the common security allocations for all users based off that profile are copied. Allocations that are not common among the users are excluded from the new group's allocations. The procedure also links all users based on the profile user to the new security group.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pdUserObj AS DECIMAL

The user's unique key field.

INPUT pcSecurityGroupName AS CHARACTER

The name of the security group to create.

Notes: None

Examples: See the processSecurity procedure in af\obj2\secusrgrp.p.

9.3.11 fieldandtokenSecurityCheck

This procedure does both a field and token security check for the specified object simultaneously, retrieving the information in one appserver hit.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pcObjectName AS CHARACTER

The current program object for security check.

INPUT pcAttributeCode AS CHARACTER

The current instance attribute posted to program.

INPUT plCheckFieldSecurity AS LOGICAL

Whether or not to extract field security.

INPUT plCheckTokenSecurity AS LOGICAL

Whether or not to extract token security.

OUTPUT pcFieldSecurity AS CHARACTER

A comma delimited list of secured fields. Each field is described by two entries in the list. The first entry is the field name qualified with the table name. The second entry is either "hidden" or "read-only", depending on the security for the particular field.

OUTPUT pcTokenSecurity AS CHARACTER

A comma delimited list of security tokens. The tokens represent objects, such as toolbar buttons and folder pages, for which the user does not have proper security clearance.

Notes: None

Examples: See the widgetWalk procedure in af\app\afsesmgrp.i.

9.3.12 fieldSecurityCheck

This procedure checks what fields a user is permitted to access.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pcObjectName AS CHARACTER

The current program object for security check.

INPUT pcAttributeCode AS CHARACTER

The current instance attribute posted to the program.

OUTPUT pcSecurityOptions AS CHARACTER

The security options as a comma-delimited list of secured fields, each with two entries.
Entry 1 = table.fieldname, Entry 2 = hidden/view.

Returns: Nothing (Procedure)

Notes: None

Examples: See the widgetWalk procedure in af\app\afsesmgrp.i.

9.3.13 fieldSecurityGet

This procedure checks fields secured for the passed-in object. If a valid procedure handle is passed in, and the object has already been secured by the Repository Manager, the security stored in the object is used. If the security is not found in the object, the applicable security is fetched from the database or AppServer by running the fieldSecurityCheck procedure.

Location: af\app\afsecmgrp.i

Parameters:

INPUT phObject AS HANDLE

The handle to the object being checked. If not specified, a standard security check is done using the object name (optional).

INPUT pcObjectName AS CHARACTER

The name of the object being checked (mandatory).

INPUT pcAttributeCode AS CHARACTER

The attribute code of the object being checked (mandatory).

OUTPUT pcSecurityOptions AS CHARACTER

The list of secured fields.

Notes: None

Examples: See the widgetWalk procedure in af\app\afsesmgrp.i.

9.3.14 getContainerIcons

This procedure returns the icons to load for each launched window.

Location: af\app\afsecmgrp.i

Parameters:

OUTPUT pcSystemIcon AS CHARACTER

The relative path to the icon.

OUTPUT pcSmallSystemIcon AS CHARACTER

The relative path to the small icon.

Notes: None

Examples: See the main block in af\sup2\aficonload.i.

9.3.15 getFieldSecurity

This procedure checks a list of field names and returns how the fields are secured.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pcFieldList AS CHARACTER

The fields to check as a CHR(1) delimited list.

OUTPUT pcSecurityList AS CHARACTER

The list of security as a comma-delimited list in the same order as the fields were passed in.

Notes: This API is meant to check field security for audit trails specifically, because the framework cannot determine from which object or container the field was updated. The API checks if the field has been secured anywhere and applies the most restrictive security if set up.

Examples: See the buildTempTable procedure in af\obj2\gstadvieww.w.

9.3.16 getMandatoryTables

This procedure checks a field to discover on what tables it is mandatory.

The procedure checks through all the locally connected databases. If it is running in an AppServer environment, it then runs itself on the AppServer checking databases connected to the AppServer.

Location: af\app\afsecmgrp.i

Parameters:

INPUT PARAMETER pcFieldName AS CHARACTER

The field to be checked.

INPUT-OUTPUT PARAMETER pcTableList AS CHARACTER

The list of tables on which the field is mandatory.

Notes: None

Examples: See the updateRecord procedure in af\obj2\gsmffvieww.w.

9.3.17 **getSecurityControl**

This procedure returns the security control details in a temp-table.

Location: af\app\afsecmgrp.i

Parameters:

OUTPUT TABLE FOR ttSecurityControl

A temp-table containing a single security control record.

Notes:

- If the temp-table is empty, the procedure first reads the details on the AppServer and populates the temp-table. If any security control settings are changed in a session, the clear cache procedure should be run so that this procedure picks up the new details.
- When running on the server, the procedure always accesses the database to get the latest security information.

Examples: See the helpContents procedure in af\app\afsesmgrp.i.

9.3.18 **menuItemSecurityCheck**

This procedure checks whether menu items (actions) are secured.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pcItem AS CHARACTER

The menu item (action) to check.

OUTPUT plItemHidden AS LOGICAL

Whether or not the item is hidden for security.

OUTPUT plItemDisabled AS LOGICAL

Whether the item is disabled, either as a result of security or because of user or menu item settings.

Notes: Any errors are returned by raising the ERROR status and placing error text in the RETURN-VALUE.

9.3.19 menuStructureSecurityCheck

This procedure checks whether menu structures (bands) are secured.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pcStructure AS CHARACTER

The menu structures (bands) to check.

OUTPUT p1StructureHidden AS LOGICAL

Whether or not the structure is hidden for security.

Notes: Any errors are returned by raising the ERROR status and placing error text in the RETURN-VALUE.

9.3.20 objectSecurityCheck

This procedure checks a user's security for the objects the user is permitted to run.

Location: af\app\afsecmgrp.i

Parameters:

INPUT-OUTPUT pcObjectName AS CHARACTER

The current program object for security check.

INPUT-OUTPUT pdObjectObj AS DECIMAL

The current instance attribute posted to program.

OUTPUT p1SecurityRestricted AS LOGICAL

The security options as a comma-delimited list of security tokens for which the user does not have security clearance.

Notes: This is currently used in toolbar panel views to disable buttons, and in folder windows to disable folder pages.

Examples: See the processRequest procedure in ry\app\ryreqsrvrp.p.

9.3.21 **plipShutdown**

This procedure is a part of the standard manager template. It runs on close of the procedure. It is available to allow customizing the framework.

Location: af\app\afsecmgrp.i

Parameters: None

Notes: This procedure currently contains no active code.

9.3.22 **rangeSecurityCheck**

This procedure checks a user's security for the passed-in range code.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pcRangeCode AS CHARACTER

The range code to check for user security clearance.

INPUT pcObjectName AS CHARACTER

The current program object for security check.

INPUT pcAttributeCode AS CHARACTER

The instance attribute posted to the program.

OUTPUT pcRangeFrom AS CHARACTER

The FROM value permitted for user, blank for all.

OUTPUT pcRangeTo AS CHARACTER

The TO value permitted for user, blank for all.

Notes: None

9.3.23 receiveCacheSessionSecurity

This procedure receives the initial session security cache.

Location: af\app\afsecmgrp.i

Parameters:

INPUT TABLE FOR ttSecurityControl

Notes: None

Examples: See the loginCacheAfter procedure in af\app\afsesmgrp.i.

9.3.24 tableSecurityCheck

This procedure checks a user's security for the table field values to which the user is permitted access.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pcOwningEntityMnemonic AS CHARACTER

The table FLA for which to check user security clearance.

INPUT pcEntityFieldName AS CHARACTER

The field name with no table prefix.

OUTPUT pcValidValues AS CHARACTER

A comma-separated list of valid values, blank for all.

Notes: None

9.3.25 tokenSecurityCheck

This procedure checks a user's security for the tokens to which the user is permitted access.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pcObjectName AS CHARACTER

The current program object for security check.

INPUT pcAttributeCode AS CHARACTER

The current instance attribute posted to the program.

OUTPUT pcSecurityOptions AS CHARACTER

The security options as a comma-delimited list of security tokens for which the user does not have security clearance.

Notes: This is currently used in toolbar panel views to disable buttons, and in folder windows to disable folder pages.

Examples: See the widgetWalk procedure in af\app\afsesmgrp.i.

9.3.26 tokenSecurityGet

This procedure checks tokens secured for the passed-in object. If a valid procedure handle is passed in, and the object has already been secured by the Repository Manager, the security stored in the object is used. If the security is not found in the object, the applicable security is fetched from the database or AppServer by running the tokenSecurityCheck procedure.

Location: af\app\afsecmgrp.i

Parameters:

INPUT phObject AS HANDLE

The handle to the object being checked. If not specified, a standard security check is done using the object name. (optional).

INPUT pcObjectName AS CHARACTER

The name of the object being checked (mandatory).

INPUT pcAttributeCode AS CHARACTER

The attribute code of the object being checked (mandatory).

OUTPUT pcSecurityOptions AS CHARACTER

The list of secured tokens and their security setting.

Notes: None

Examples: See the widgetWalk procedure in af\app\afsesmgrp.i.

9.3.27 updateUserAllocations

This procedure creates user security allocations in the Repository.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pdUserObj AS DECIMAL

The user obj number or 0 for all.

INPUT pdOrganisationObj AS DECIMAL

The login organization obj or 0 for all.

INPUT TABLE FOR ttUpdatedAllocations

Temp-table of user allocations to update.

Notes: None

Examples: See the updateForAll procedure in af\obj2\grpcomsupr.p.

9.3.28 userLoginOrganisations

This procedure checks which organizations a user can access.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pdUserObj AS DECIMAL

The user_obj.

OUTPUT pcOrganisations AS CHARACTER

A comma-delimited list of list pairs. The organization obj is the first entry, and the name of the organization is the second entry.

Notes: This procedure is included to support backward compatibility only.

9.3.29 userSecurityCheck

This procedure checks a user's security allocation for the passed-in company and security option. The types of security checks that could be made include checks for access to tokens, fields, data, data ranges, and menu items. The type of security check depends on the passed-in entity mnemonic and obj.

Location: af\app\afsecmgrp.i

Parameters:

INPUT pdUserObj AS DECIMAL

The user being checked.

INPUT pdOrganisationObj AS DECIMAL

The company the user is logged into.

INPUT pcOwningEntityMnemonic AS CHARACTER

The security table being checked.

INPUT pdOwningObj AS DECIMAL

The security table object being checked.

INPUT p1ReturnValues AS LOGICAL

Whether or not to return security values.

OUTPUT plSecurityRestricted AS LOGICAL

Returns YES if security check is passed.

OUTPUT pcSecurityValue1 AS CHARACTER

Returns any specific security data.

OUTPUT pcSecurityValue2 AS CHARACTER

Returns any specific security data.

Notes: None

Examples: See the main block in af\app\afchkuserp.p.

Session Manager

The Session Manager handles starting and stopping application components in a distributed environment.

This chapter covers the following subjects:

- [Overview](#)
- [Temp-tables used by Session Manager](#)
- [Session Manager APIs](#)

10.1 Overview

The Session Manager controls the starting and stopping of application components in a distributed environment. It passes client requests to execute server-side business logic to the appropriate procedures, wherever they are running. It also manages client context for the stateless execution of all server-side components. It also supports the following application services:

- Context management
- Persistent procedure management
- Session property management
- Error and message handling
- Online help

After all the necessary connections have been established by the Connection Manager, the Session Manager starts to handle the actual running of the session. It is responsible for coordinating the activities of the various other Managers used in the session.

In a distributed environment, the Session Manager is started on both the client and server sides. There is a wrapper program for each side that either sets the global variable, `server-side`, to YES for the server-side version or to NO for the client-side version. Both programs then include the main code of the manager from the `af\app\afsesmgrp.i` include file. The differences in how the manager operates on each side are triggered by checking the value of the `server-side` variable.

The Session Manager's handle is stored in the global shared variable, `gshSessionManager`.

Table 10–1 shows the files that contain the Session Manager’s code.

Table 10–1: Session Manager files

Type	Path
Client-side wrapper	af\sup2\afsecclntp.p
Server-side wrapper	af\app\afsessrvp.p
Main code	af\app\afsesmgrp.i
Included files	adecomm\appsrvtt.i adecomm\appserv.i adm\method\attribut.i adm2\calltables.i adm2\callttparam.i adm2\globals.i adm2\ttaction.i adm2\tttoolbar.i af\app\afsesgetglcp.i af\app\afittglobalctrl.i af\app\afittpersist.i af\app\afittprofiledata.i af\app\afittsecurityctrl.i af\app\afitttranslate.i af\app\afitttranslation.i af\app\gsttenmn.i af\app\logintt.i af\sup\windows.i af\sup2\afcheckerr.i af\sup2\aferrortxt.i af\sup2\aflaunch.i af\sup2\afittcombo.i ry\app\rydefrescd.i

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in the *Progress Dynamics Programming Handbook*.

10.2 Temp-tables used by Session Manager

Because the Session Manager coordinates so much of what is happening in the Progress Dynamics framework, it makes extensive use of temp-tables. It references several temp-tables defined elsewhere in the Progress ADE, including those in the following files:

- adecomm\appsrvtt.i
- adm2\calltables.i
- adm2\callttparam.i
- adm2\logintt.i
- adm2\ttaction.i
- adm2\tttoolbar.i
- adm2\tttranslate.i

The Session Manager uses the temp-tables described in [Table 10–2](#) to cache session and context management information.

Table 10–2: Temp-tables defined in aflapp\afsesmgrp.i

Temp-table	Fields (data types)
ttProperty	propertyName (Character) propertyValue (Character)
ttPersistProc	hProc (Handle)
ttActionUnderway	action_underway_origin (Character) action_table_flg (Character) action_type (Character) action_primary_key (Character) action_scm_object_name (Character)
ttViewerCol	dColumn (Decimal) dColWidth (Decimal) dMaxLabel (Decimal) dNewCol (Decimal)

The Session Manager uses the temp-table described in [Table 10–3](#) to cache user login information.

Table 10–3: Temp-table defined in a\app\login.tti

Temp-table	Fields (data types)
ttLoginUser	encoded_user_name (Character) default_organisation_obj (Decimal) language_obj (Decimal)

The Session Manager uses the temp-tables described in [Table 10–4](#) to cache information about persistently running procedures in current session.

Table 10–4: Temp-tables defined in a\app\lfttpersist.i

Temp-table	Fields (data types)
ttPersistentProc	physicalName (Character) logicalName (Character) runAttribute (Character) childDataKey (Character) onAppserver (Logical) procedureType (Character) procedureHandle (Handle) runPermanent (Logical) multiInstanceSupported (Logical) currentOperation (Character) uniqueId (Integer) startDate (Date) startTime (Integer) procedureVersion (Character) procedureNarration (Character) internalEntries (Character)
ttProcDependency	parentProcedureHandle (Handle) childProcedureHandle (Handle) parentType (Character) childType (Character)

The Session Manager uses the temp-table described in [Table 10–5](#), a copy of the Repository gsc_global_control table, to cache information about system-wide defaults.

Table 10–5: Temp-table defined in a\app\af\ftglobalctrl.i

Temp-table	Fields (data types)
ttGlobalControl	global_control_obj (Decimal) default_country_obj (Decimal) default_nationality_obj (Decimal) default_language_obj (Decimal) default_currency_obj (Decimal) date_format (Character) date_format_mask (Character) repository_site_number (Integer)

Because the Session Manager coordinates the activities of all the other Managers once a session has started, it makes use of most of the other Managers’ temp-tables. [Table 10–6](#) lists these temp-tables and where they are described in this reference manual.

Table 10–6: Other temp-tables used by the Session Manager

Temp-table	Described in
ttEntityMnemonic	“Temp-tables used by General Manager”
ttEntityDisplayedField	“Temp-tables used by General Manager”
ttEntityMap	“Temp-tables used by General Manager”
ttProfileData	“Temp-tables used by Profile Manager”
ttSecurityControl	“Temp-Tables used by Security Manager”
ttTranslation	“Temp-tables used by Localization Manager”
ttUser	“Temp-tables used by General Manager”

10.3 Session Manager APIs

This section lists the APIs that you can use with the Session Manager.

10.3.1 activateSession

This procedure sets up the record for a remote session in the Repository's `gst_session` table.

Location: `af\app\afsesmgrp.i`

Parameters:

INPUT `pcOldSessionID` AS CHARACTER

INPUT `pcNewSessionID` AS CHARACTER

INPUT `pcSessType` AS CHARACTER

INPUT `pcNumFormat` AS CHARACTER

INPUT `pcDateFormat` AS CHARACTER

INPUT `plReactivate` AS LOGICAL

INPUT `plConfirmExpired` AS LOGICAL

Notes: None

Examples: See the main block in `icf\as_activate.p`.

10.3.2 askQuestion

This procedure handles the display of all question message types and supports any button combination. The default button list is "OK,CANCEL". If OK is passed in, the default label to return is OK. Otherwise, the label is the first button in the list. If available, the default cancel button is CANCEL. Otherwise, the first entry in the list is used. The default title is "Question".

If running server-side, the messages cannot be displayed and must be written to the message log. Because there is no user interface on the server side, the procedure always returns the default button label and answer.

If running client-side, the messages are displayed in a dialog window. The procedure checks the property "suppressDisplay" in the Session Manager. If the property is set to YES, the procedure does not display the message. It passes the message to the log. This is useful when running take-on procedures on the client side.

The messages are passed to the procedure, `af\app\afmessagep.p`, on the AppServer for interpretation. This procedure performs the necessary tasks to return the formatted messages. This might include the following tasks:

- Formatting the messages appropriately
- Reading the text from the Progress Dynamics message file
- Interpreting the carrot-delimited lists that come back from triggers
- Dealing with ADM2 CHR(4)-delimited messages
- Translating the message if required

Once the messages are formatted, a client-side message displays using the standard Progress Dynamics message dialog. On the server-side, the Progress Dynamics error log is updated with the error and, if possible, an e-mail is sent to the currently logged-in user to notify them. This process also occurs if the error log flag is set to YES or message display suppression is enabled.

Location: `af\app\afsesmgrp.i`

Parameters:

INPUT `pcMessageList` AS CHARACTER

INPUT `pcButtonList` AS CHARACTER

INPUT `pcDefaultButton` AS CHARACTER

INPUT `pcCancelButton` AS CHARACTER

INPUT `pcMessageTitle` AS CHARACTER

INPUT `pcDataType` AS CHARACTER

INPUT `pcFormat` AS CHARACTER

INPUT-OUTPUT `pcAnswer` AS CHARACTER

OUTPUT `pcButtonPressed` AS CHARACTER

Notes: When running client-side, the procedure returns the untranslated button text of the pressed button. When running server-side, the untranslated button text of the default button is returned.

Examples: See the `logMessage` procedure in `af\cod2\fullobjectw.w`.

10.3.3 contextHelp

This procedure serves as the context help launcher to aid in the Progress Dynamics context help integration.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phObject AS HANDLE

The handle of the object containing widget (THIS-PROCEDURE).

INPUT phWidget AS HANDLE

The handle of the widget that has focus (FOCUS).

Notes: There is an event in visualcustom.i that runs this procedure on a help event anywhere in the frame.

Examples: See the main block in af\cod2\gstrvdi1gd.w.

10.3.4 createLinks

This procedure aids in creating links between objects.

Location: af\app\afsesmgrp.i

Parameters:

INPUT pcPhysicalName AS CHARACTER

INPUT phProcedureHandle AS HANDLE

INPUT phObjectProcedure AS HANDLE

INPUT plAlreadyRunning AS LOGICAL

Notes: None

Examples: See the launchContainer procedure in af\app\afsesmgrp.i.

10.3.5 deletePersistentProc

This procedure deletes persistent procedures started since the Session Manager started.

Location: af\app\afsesmgrp.i

Parameters: None

Notes: None

Examples: See the plipShutdown procedure in af\app\afsesmgrp.i.

10.3.6 fixQueryString

This function checks for nondecimal point decimal places in the query string and replaces them with full stops to resolve issues when running with non-American numeric formats selected.

Before a query prepare is used, call this procedure first to resolve any issues in the query string, such as decimal formatting. This is most important when the query string contains stringed decimal values.

Location: af\app\afsesmgrp.i

Parameters:

INPUT pcQueryString AS CHARACTER

The query to be checked.

Returns: CHARACTER

Notes: None

Examples: See the checkIfOverlaps procedure in af\app\afgenmgrp.i.

10.3.7 getActionUnderway

This procedure gets the values for fields in the ttActionUnderway temp-table for the passed-in record.

Location: af\app\afsesmgrp.i

Parameters:

INPUT pcActionUnderwayOrigin AS CHARACTER

INPUT pcActionType AS CHARACTER

INPUT pcActionScmObjectName AS CHARACTER

INPUT pcActionTablePrimaryFla AS CHARACTER

INPUT pcActionPrimaryKeyValues AS CHARACTER

INPUT plActionUnderwayRemove AS LOGICAL

OUTPUT plActionUnderway AS LOGICAL

Notes: None

Examples: See the createNewSDO procedure in af\cod2\fullloobjcw.w.

10.3.8 getCurrentLogicalName

This function returns the value of the gcLogicalContainerName variable set in the Session Manager's launchContainer procedure.

Location: af\app\afsesmgrp.i

Parameters: None

Returns: CHARACTER

Notes: None

10.3.9 getGlobalControl

This procedure returns the global control details in the form of a temp-table, ttGlobalControl. If the temp-table is empty, then it first goes to the AppServer to read the details and populate the temp-table. On the server, the database is always accessed to get the information.

Location: af\app\afsesmgrp.i

Parameters:

OUTPUT TABLE FOR ttGlobalControl

Temp-table containing a single latest global control record.

Notes: None

Examples: See the getGlobalControlObj function in af\obj2\gscddimprv.w.

10.3.10 **getHelp**

This procedure returns a temp-table of widgets for a passed-in object. It populates the temp-table with the help context already stored on the database.

Location: af\app\afsesmgrp.i

Parameters:

INPUT-OUTPUT TABLE-HANDLE hHelpTable

Notes: None

Examples: See the buildTempTable procedure in ry\obj\rydynhelpv.w.

10.3.11 **getInternalEntryExists**

This function checks whether a procedure or function exists for a given handle.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phProcedure AS HANDLE

INPUT pcProcedureName AS CHARACTER

Returns: LOGICAL

Notes:

- If the procedure handle is a proxy handle for a persistent procedure running remotely in the context of a Progress AppServer, the :PROXY attribute is TRUE. If not, it is FALSE.
- This function is used because you cannot read the :INTERNAL-ENTRIES attribute of procedures that are running remotely.

Examples: See the killPlips procedure in af\app\afsesmgrp.i.

10.3.12 **getLoginUserInfo**

This procedure returns user information used by the login process. It returns the encoded user name, the default login company, and the default language.

Location: af\app\afsesmgrp.i

Parameters:

OUTPUT TABLE FOR ttLoginUser

Notes: None

Examples: See the main block in af\app\cache\login.p.

10.3.13 getPersistentProcs

This procedure retrieves a temp-table, ttPersistentProc, of running persistent procedures. The temp-table is used to supply data for display in a browser outside the Session Manager.

Location: af\app\afsesmgrp.i

Parameters:

OUTPUT TABLE FOR ttPersistentProc

The temp-table of persistent procedures.

Notes: None

10.3.14 getPropertyList

This function retrieves properties from the local temp-table if available. Otherwise, it retrieves them from the context in the Repository through the server-side Session Manager.

The function returns a CHR(3)-delimited list of corresponding property values.

Location: af\app\afsesmgrp.i

Parameters:

INPUT pcPropertyList AS CHARACTER

A comma-delimited list of properties for which to retrieve values.

INPUT plSessionOnly AS LOGICAL

If running client-side and this parameter is set to YES, then the Repository is not checked.

Returns: CHARACTER

Notes:

- The local cache temp-table is only checked when running client-side.
- If a server-side routine is not connected to the AppServer and is running client-side instead, the function does not check the context in the Repository. All properties should be set in the local temp-table in this case.

Examples: See the askQuestion procedure in af\app\afsesmgrp.i.

10.3.15 helpAbout

This procedure displays the “About” help message. The procedure uses an ADM2 showMessage call using the passed-in container so that all the object names and versions are shown in the system information.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phContainer AS HANDLE

The container procedure handle.

Notes: None

Examples: See the buildMenus procedure in af\cod2\afmenumaintw.w.

10.3.16 helpContents

This procedure displays the help contents from the help file.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phContainer AS HANDLE

The container procedure handle.

Notes: None

Examples: See the buildMenus procedure in af\cod2\afmenumaintw.w.

10.3.17 helpHelp

This procedure displays the help contents from the help file.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phContainer AS HANDLE

The container procedure handle.

Notes: None

Examples: See the buildMenus procedure in af\cod2\afmenumaintw.w.

10.3.18 helpTopics

This procedure displays the help contents from the help file.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phContainer AS HANDLE

The container procedure handle.

Notes: None

Examples: See the buildMenus procedure in af\cod2\afmenumaintw.w.

10.3.19 htmlHelpKeywords

This procedure displays the help topics using keyword lookup from a HTML help file.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phParent AS HANDLE

The parent handle (frame) or the unknown value (?).

INPUT pcHelpFile AS CHARACTER

The help file.

INPUT pcHelpKeywords AS CHARACTER

The help keywords.

Notes: None

10.3.20 **htmlHelpTopic**

This procedure displays a specific help topic in a HTML help file.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phParent AS HANDLE

The parent handle (frame) or the unknown value (?).

INPUT pcHelpFile AS CHARACTER

The help file.

INPUT pcHelpTopic AS CHARACTER

The help topic.

Notes: None

Examples: See the helpContents procedure in af\app\afsesmgrp.i.

10.3.21 **increaseFrameforPopup**

This procedure increases the width of a frame, and window if neccessary, for a popup.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phObject AS HANDLE

INPUT phFrame AS HANDLE

INPUT phLookup AS HANDLE

INPUT phWidget AS HANDLE

Notes: Replaces increaseFrameWidth procedure.

10.3.22 **isObjQuoted**

This function checks the object number in the query string to determine whether or not it is wrapped in quotes. This is important when an application runs in European numerical format, because the object number must be treated differently when it is enclosed in quotes.

When running with European numerical format, an object number enclosed in quotes must retain the comma inside the quotes in order to convert properly. If it is not enclosed in quotes, then the comma is replaced with a decimal point to ensure that the query resolves properly.

This function is called by the `fixQueryString` procedure to determine whether or not the object number is quoted. If necessary, that procedure replaces the object number's decimal separator.

Location: `af\app\afsesmgrp.i`

Parameters:

INPUT `pcQueryString` AS CHARACTER

The query being examined.

INPUT `piPosition` AS INTEGER

Returns: LOGICAL

Notes: None

Examples: See the `fixQueryString` procedure in `af\app\afsesmgrp.i`.

10.3.23 killPlips

This procedure shuts down PLIPs cleanly, forcing the correct update of the running procedure's temp-table.

Location: `af\app\afsesmgrp.i`

Parameters:

INPUT `pcPlipNames` AS CHARACTER

A CHR(3)-delimited list of PLIP names to kill.

INPUT `pcPlipHandles` AS CHARACTER

A CHR(3)-delimited list of PLIP handles to kill.

Notes:

- Only one of the parameters is required, depending on whether the PLIP name or the PLIP handle is known. A combination can be passed in if required.
- If PLIP names are used, the full PLIP name including relative path and .p extension must be specified, as it was specified when the PLIP was launched.

Examples: See the `PlipShutdown` procedure in `ry\app\rycusrmgrp.i`.

10.3.24 killProcedure

This procedure removes a procedure from the temp-table of running procedures, ttPersistentProc. This is used to remove all types of procedures from the temp-table, as launched by the launchContainer and launchProcedure procedures.

Location: af\app\afsesmgrp.i

Parameters:

INPUT pcPhysicalName AS CHARACTER

The physical object filename, with path and extension.

INPUT pcLogicalName AS CHARACTER

The logical object name, if applicable and known.

INPUT pcChildDataKey AS CHARACTER

The child data key, if applicable.

INPUT pcRunAttribute AS CHARACTER

The run attribute, if required to post into the container run.

INPUT p10nAppserver AS LOGICAL

If running on the AppServer, set to YES.

Notes: None

Examples: See the main block in ry\app\ryplipshut.i.

10.3.25 launchContainer

This procedure launches a Progress Dynamics container object. It determines if the object is already running and whether the existing instance should be replaced or a new instance run. The procedure then updates the temp-table of running persistent procedures, ttPersistentProc, with the appropriate details.

Location: af\app\afsesmgrp.i

Parameters:

INPUT pcObjectFileName AS CHARACTER

The object filename. If it is not known, the physical or logical names.

INPUT pcPhysicalName AS CHARACTER

The physical object name, with path and extension, if known.

INPUT pcLogicalName AS CHARACTER

The logical object name, if applicable and known.

INPUT p1OnceOnly AS LOGICAL

If YES, then the procedure checks for an already running instance and uses it if possible.

INPUT pcInstanceAttributes AS CHARACTER

The instance attributes to pass to the container.

INPUT pcChildDataKey AS CHARACTER

The child data key, if applicable.

INPUT pcRunAttribute AS CHARACTER

The run attribute, if required to post into the container run.

INPUT pcContainerMode AS CHARACTER

The container mode, for example, modify, view, add, or copy.

INPUT phParentWindow AS HANDLE

The parent (caller) window handle, if known. (Container window handle.)

INPUT phParentProcedure AS HANDLE

The parent (caller) procedure handle, if known. (Container procedure handle.)

INPUT phObjectProcedure AS HANDLE

The parent (caller) object handle, if known. This is the handle at the end of the toolbar link, for example, a browser.

OUTPUT phProcedureHandle AS HANDLE

The procedure handle of object run.

OUTPUT pcProcedureType AS CHARACTER

The procedure type, for example, ADM1, ADM2, ICF, or "".

Notes: None

Examples: See the main block in `af\app\xmlcfgp.p`.

10.3.26 LaunchExternalProcess

This procedure launches an external process.

Location: `af\app\afsesmgrp.i`

Parameters:

INPUT `pcCommandLine` AS CHARACTER

The command line, for example, “notepad.exe”.

INPUT `pcCurrentDirectory` AS CHARACTER

The default directory for the process.

INPUT `piShowWindow` AS INTEGER

The show window flag accepts the following values:

- 0 (Hidden)
- 1 (Normal)
- 2 (Minimized)
- 3 (Maximized)

OUTPUT `piResult` AS INTEGER

Returns 0 if it fails, or the handle of the new process.

Notes: This procedure uses the `createProcess` function from `af\sup\windows.i`.

10.3.27 launchProcedure

This procedure launches a business logic procedure or manager procedure. It determines if the procedure is already running and whether the existing instance should be replaced or a new instance run. If required, the procedure also connects to the AppServer partition. The procedure then updates the temp-table of running persistent procedures with the appropriate details.

Location: `af\app\afsesmgrp.i`

Parameters:

INPUT pcPhysicalName AS CHARACTER

The physical object filename, with path and extension.

INPUT p1OnceOnly AS LOGICAL

If YES, then the procedure checks for an already running instance and uses it, if possible.

INPUT pcOnAppserver AS CHARACTER

Accepts values of YES, NO, or APPSERVER.

INPUT pcAppserverPartition AS CHARACTER

The AppServer partition name to run on.

INPUT p1RunPermanent AS LOGICAL

The default is NO.

OUTPUT phProcedureHandle AS HANDLE

The procedure handle of object run.

Notes:

- If p1RunPermanent is YES, the procedure is not automatically killed when an AppServer agent deactivates. Ordinarily this flag should be NO, and the deactivation routine should delete all running procedures at the end of an AppServer request. When procedures are closed down correctly, they are removed from the temp-table and deleted. This behavior tidies up any procedures started outside of this control procedure or shutdown incorrectly for some reason.
- If pcOnAppserver is APPSERVER, the procedure can only be run on AppServer. If the flag is YES and no AppServer partition is given, the default is AstraAppServer and the gshAstraAppserver session handle is used. If another partition is given and this flag is not NO, the partition is connected if required. Any partitions connected in this manner are disconnected by the shutdown procedure, af\sup2\afshutdwnp.p.
- This procedure is not normally required for Progress Dynamics Managers whose handles are available through the system wide global shared variables. However, this procedure is used for those Managers when they first run to add the Managers to the temp-table of running persistent procedures.

Examples: See the launchClassObject function in ry\app\ryrepmgrp.i.

10.3.28 notifyUser

This procedure notifies the user of some message by some means, for example, e-mail.

Location: af\app\afsesmgrp.i

Parameters:

INPUT pdUserObj AS DECIMAL

The object number of the user record to notify.

INPUT pcUserName AS CHARACTER

The user name of user record to notify. This is used only when the pdUserObj is 0.

INPUT pcAction AS CHARACTER

The action used to notify the user, for example, “email”.

INPUT pcSubject AS CHARACTER

The subject of the message.

INPUT pcMessage AS CHARACTER

The message text.

OUTPUT pcFailedReason AS CHARACTER

The reason for any failure.

Notes: None

Examples: See the askQuestion procedure in af\app\afsesmgrp.i.

10.3.29 plipShutdown

This procedure is a part of the standard Progress Dynamics Manager template. On close of the procedure, it runs the deletePersistentProc procedure.

You should not directly launch this procedure. The Session Manager should only be shutdown by the Configuration File Manager’s sessionShutdown procedure. However, for any custom tasks that occur when the Session Manager shuts down, you can add code to this procedure.

Location: af\app\afsesmgrp.i

Parameters: None

Notes: None

10.3.30 **resizeLookupFrame**

This procedure resizes a lookup SDF frame to fit new labels.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phObject AS HANDLE

The object handle.

INPUT phFrame AS HANDLE

The SDF frame handle.

INPUT pdAddCol AS DECIMAL

The number to add to all columns.

Notes: None

Examples: See the translateWidgets procedure in af\app\afsesmgrp.i.

10.3.31 **resizeNormalFrame**

This procedure resizes a standard frame to fit new labels. SDF frames do not use this procedure.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phObject AS HANDLE

The object handle.

INPUT phFrame AS HANDLE

The frame handle.

INPUT pdNewWidth AS DECIMAL

The new column width.

Notes: None

Examples: See the translateWidgets procedure in af\app\afsesmgrp.i.

10.3.32 **resizeSDFFrame**

This procedure resizes a SDF frame to fit new labels.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phObject AS HANDLE

 The object handle.

INPUT phFrame AS HANDLE

 The frame handle.

INPUT pdAddCol AS DECIMAL

 The number to add to all columns.

Notes: None

Examples: See the translateWidgets procedure in af\app\afsesmgrp.i.

10.3.33 **runLookup**

This procedure launches a lookup window for a widget. If the data type is a date, then a pop-up calendar is displayed. If the data type is an integer or decimal, then a pop-up calculator is displayed.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phFocus AS HANDLE

 The handle of the focused widget.

Notes: None

Examples: See the widgetWalk procedure in af\app\afsesmgrp.i.

10.3.34 sendEmail

This procedure sends an e-mail message. This procedure is similar to notifyUser, but is more flexible and has additional options specific to sending an e-mail message. Most of the parameters are optional and can be left blank, as appropriate. The procedure allows multiple file attachments to be sent using comma-delimited lists. Because this procedure uses MAPI for client e-mail, it works with whatever e-mail is installed on the client PC sending the e-mail. On the server, it uses sendmail, and some options might not be supported.

Location: af\app\afsesmgrp.i

Parameters:

INPUT cEmailProfile AS CHARACTER

The mail profile to use, for example, Microsoft Outlook.

INPUT cToEmail AS CHARACTER

A comma-delimited list of e-mail addresses to which the message is sent.

INPUT cCcEmail AS CHARACTER

A comma-delimited list of e-mail addresses to which the message is copied.

INPUT cSubject AS CHARACTER

The subject of the message.

INPUT cMessage AS CHARACTER

The message text.

INPUT cAttachmentName AS CHARACTER

A comma-delimited list of attachment filenames.

INPUT cAttachmentFPath AS CHARACTER

A comma-delimited list of attachment filenames with full path.

INPUT lDisplayDialog AS LOGICAL

If set to YES, the message is displayed in a dialog box for modification before sending. If set to NO, then the message is sent immediately.

INPUT iImportance AS INTEGER

The accepted values are: 0 = low, 1 = medium, 2 = high.

INPUT `lReadReceipt` AS LOGICAL

YES = return a read receipt.

INPUT `lDeliveryReceipt` AS LOGICAL

YES = return a delivery receipt.

INPUT `cOptions` AS CHARACTER

Not currently used, set aside for future settings. It can contain a comma-delimited list of setting-value pairs of other settings.

OUTPUT `cFailedReason` AS CHARACTER

If the procedure failed, returns the reason why. Otherwise, returns blank, "".

Notes: None

Examples: See the `notifyUser` procedure in `af\app\afsesmgrp.i`.

10.3.35 `setActionUnderway`

This procedure sets the values in the `ttActionUnderway` temp-table for the passed-in record.

Location: `af\app\afsesmgrp.i`

Parameters:

INPUT `pcActionUnderwayOrigin` AS CHARACTER

SCM or DYN.

INPUT `pcActionType` AS CHARACTER

ASS, DEL, MOV, or ADD.

INPUT `pcActionScmObjectName` AS CHARACTER

INPUT `pcActionTablePrimaryFla` AS CHARACTER

INPUT `pcActionPrimaryKeyValues` AS CHARACTER

Notes: None

Examples: See the `createNewSDO` procedure in `af\cod2\full0objcw.w`.

10.3.36 setAttributesInObject

This procedure sets instance attributes in an object. It is run from the launch container to pass instance attributes into an object.

The instance attribute list is in the same format as returned to the instancePropertyList function:

1. CHR(3) between entries
2. CHR(4) between the property name and its value in each entry.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phObject AS HANDLE

The object handle.

INPUT pcPropList AS CHARACTER

The instance attribute list.

Notes: None

Examples: See the launchContainer procedure in af\app\afsesmgrp.i.

10.3.37 setPropertyList

This function sets properties in the local temp-table, ttProperty, if available. It then uses the server-side Session Manager procedure to set the properties in the context database.

Location: af\app\afsesmgrp.i

Parameters:

INPUT pcPropertyList AS CHARACTER

A comma-delimited list of property names for which to set values.

INPUT pcPropertyValues AS CHARACTER

A CHR(3)-delimited list of corresponding property values.

INPUT plSessionOnly AS LOGICAL

If set to YES, the function creates only a local temp-table record.

Returns: LOGICAL

Notes: None

Examples: See the relogin procedure in af\app\afsesmgrp.i.

10.3.38 setReturnValue

This procedure returns whatever was sent in to set the required RETURN-VALUE.

Location: af\app\afsesmgrp.i

Parameters:

INPUT pcReturnValue AS CHARACTER

The required return value.

Notes: None

10.3.39 setSecurityForDynObjects

This function sets security properties for Dynamic Lookups and Dynamic Combos.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phWidget AS HANDLE

INPUT pcSecuredFields AS CHARACTER

INPUT pcDisplayedFields AS CHARACTER

INPUT pcFieldSecurity AS CHARACTER

INPUT phViewer AS HANDLE

Returns: CHARACTER

Notes: None

Examples: See the widgetWalk procedure in af\app\afsesmgrp.i.

10.3.40 showMessages

This is the central procedure for the displaying all message types including Message (MES), Information (INF), Warnings (WAR), Errors (ERR), Serious Halt Errors (HAL), and About Window (ABO). Any button combination is supported.

The default message type is ERR. The default button list is OK. The default label to return is OK, if OK exists. Otherwise, the default is the first button in the list. The default cancel button is also OK, or the first entry in the button list. The default title depends on the message type.

If running server-side, the messages cannot be displayed and can only be written to the message log. Because there is no user interface on the server side, the default button label is returned.

If running client-side, the messages are displayed in a dialog window. The procedure checks the property “suppressDisplay” in the Session Manager. If it is set to YES, the procedure does not display the message. The message is passed to the log. This is useful when running take-on procedures client side.

The messages are passed to the procedure, `af\app\afmessagep.p`, on the AppServer for interpretation. This procedure performs the necessary tasks to return the formatted messages. This might include the following tasks:

- Formatting the messages appropriately
- Reading the text from the Dynamics message file
- Interpreting the carrot-delimited lists that come back from triggers
- Dealing with ADM2 CHR(4)-delimited messages
- Translating the message if required

Once the messages have been formatted, a client-side message is displayed using the standard Dynamics message dialog, `af\cod2\afmessaging.w`. On the server-side, the Dynamics error log is updated with the error and, if possible, an e-mail is sent to the currently logged-in user to notify them of the error. This process is also followed if the error log flag is set to YES or message display suppression is enabled.

Location: `af\app\afsesmgrp.i`

Parameters:

INPUT pcMessageList AS CHARACTER

INPUT pcMessageType AS CHARACTER

INPUT pcButtonList AS CHARACTER

INPUT pcDefaultButton AS CHARACTER

INPUT pcCancelButton AS CHARACTER

INPUT pcMessageTitle AS CHARACTER

INPUT plDisplayEmpty AS LOGICAL

INPUT phContainer AS HANDLE

OUTPUT pcButtonPressed AS CHARACTER

Notes: When running client-side, the procedure returns the untranslated button text of the pressed button. When running server-side, the untranslated button text of the default button is returned.

Examples: See the main block in af\cod2\aftemlognw.w.

10.3.41 showWarningMessages

This procedure issues a warning to a user without generating an input-blocking statement in the process.

Location: af\app\afsesmgrp.i

Parameters:

INPUT pcMessageList AS CHARACTER

A message using the standard formatting from aferrortxt.i. This parameter can contain several messages.

INPUT pcMessageType AS CHARACTER

For example, ERR (Error) or INF (Information).

INPUT pcMessageTitle AS CHARACTER

The title of the message dialog.

Notes: None

10.3.42 translateWidgets

This procedure translates widget labels. It is called from the widgetWalk procedure.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phObject AS HANDLE

The object handle.

INPUT phFrame AS HANDLE

The frame handle.

INPUT TABLE FOR ttTranslate

Notes: None

Examples: See the widgetWalk procedure in af\app\afsesmgrp.i.

10.3.43 updateErrorLog

This procedure updates the messages into the error log database table.

Location: af\app\afsesmgrp.i

Parameters:

INPUT pcSummaryList AS CHARACTER

A CHR(3)-delimited list of summary messages.

INPUT pcFullList AS CHARACTER

A CHR(3)-delimited list of full messages.

Notes: None

Examples: See the askQuestion and showMessages procedures in af\app\afsesmgrp.i.

10.3.44 updateHelp

This procedure updates the help table with the supplied temp-table.

Location: af\app\afsesmgrp.i

Parameters:

INPUT TABLE-HANDLE hHelpTable

Notes: None

Examples: See the updateRecord procedure in ry\obj\rydynhelpv.w.

10.3.45 widgetWalk

This procedure walks the widget tree for the passed-in frame.

Location: af\app\afsesmgrp.i

Parameters:

INPUT phContainer AS HANDLE

The container handle.

INPUT phObject AS HANDLE

The object handle.

INPUT phFrame AS HANDLE

The frame or window handle.

INPUT pcAction AS CHARACTER

The action code, for example, setup.

INPUT p1PopupsInFields AS LOGICAL

Determines where calendar and calculator pop-up buttons are placed. If YES, the button is placed inside the field on the right-hand side. If NO, extra space is made for the button, and the button is placed in this extra space.

Notes: This procedure no longer appends the pop-up handle to the widget's PRIVATE-DATA for static objects.

User Interface Manager

The User Interface Manager handles the processing need to create and return the DHTML to fulfill client requests received by the Web Request Manager.

This chapter covers the following subjects:

- [Overview](#)
- [User Interface Manager APIs](#)

11.1 Overview

The User Interface (UI) Manager generates the DHTML to satisfy a client request received by the Web Request Manager. The UI Manager is a server-side component of the framework. It delivers all user interface and application data to the client.

The UI Manager retrieves the UI information from the Repository and other managers. The UI Manager then fetches the necessary application data from the data sources like SDOs and SBOs. To reduce network traffic, the UI and application data are sent to the client as part of the same response. The UI information is sent to the client only once per session. While in comparison, there might be multiple data requests for a container each session because of data navigation.

This manager should only be accessed through the `escapeData` and `setClientAction` procedures.

You can retrieve the handle of the UI Manager by running code like the following example:

```
ASSIGN hUserInterfaceManager =  
DYNAMIC-FUNCTION("getManagerHandle":U, INPUT "UserInterfaceManager":U).
```

Table 11–1 shows the files that contain the User Interface Manager’s code.

Table 11–1: User Interface Manager files

Type	Path
Client-side wrapper	NA
Server-side wrapper	NA
Main code	ry\app\ryuimsrvp.p
Included files	adm2\treettdef.i adm2\ttaction.i adm2\tttoolbar.i af\sup\afghplipdf.i af\sup2\afglobals.i ry\app\rydefrescd.i ry\app\ryobjretri.i ry\app\ryuimbl.i ry\inc\lval.i web2\wrap-cgi.i wrappers\lognote.i
Other important files	ry\app\ryuimbl.i

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in the *Progress Dynamics Programming Handbook*.

11.2 User Interface Manager APIs

This section lists the APIs that you can use with the User Interface Manager.

11.2.1 **escapeData**

This function will escape data line single quotes, new line, and carriage return characters.

Location: ry\app\ryuimb1.i

Parameters:

pcData AS CHARACTER

Returns: CHARACTER

Notes: None

Examples: See the ouputConflictData function in ry\app\ryuimb1.i.

11.2.2 **setClientAction**

This procedure applies changes to the DHTML client in response to server-side business logic. Client actions are queued during a web request and sent together to the DHTML client at the end of the response stream.

The action is passed with the following syntax:

<dataobject>.<field name>.<action name>

The field name is not required for all actions.

Location: ry\app\ryuimb1.i

Parameters:

INPUT cAction AS CHARACTER

The action to apply.

Notes: None

Examples: See the setMessage procedure in ry\app\ryuimb1.i.

Web Request Manager

The Web Request Manager is the single point of entry for all web requests.

This chapter covers the following subjects:

- [Overview](#)
- [Web Request Manager APIs](#)

12.1 Overview

The Web Request Manager is the single point of entry for all web requests. It then calls the other Dynamics managers that apply security, generate or retrieve session information, interact with databases, and create the DHTML.

This manager should only be accessed through the processRequest procedure.

The Web Request Manager's handle is stored in the global shared variable, gshWebManager.

[Table 12-1](#) shows the files that contain the Web Request Manager's code.

Table 12-1: Web Request Manager files

Type	Path
Client-side wrapper	NA
Server-side wrapper	NA
Main code	ry\app\ryreqsrvrp.p
Included files	af\app\afttsecurityctrl.i af\sup\afghplipdf.i af\sup2\afglobals.i web2\wrap-cgi.i wrappers\lognote.i

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in the [Progress Dynamics Programming Handbook](#).

12.2 Web Request Manager APIs

This section lists the APIs that you can use with the Web Request Manager.

12.2.1 processRequest

This procedure processes a request based on the client type.

Location: ry\app\ryreqsrvrp.p

Parameters:

INPUT pcAppProgram AS CHARACTER

The repository object name.

Notes: None

Index

A

activateSession procedure
 Session Manager 10–7

AppServer Connection Manager

- appServerConnect procedure 2–11
- appServerDisconnect procedure 2–11
- connectService procedure 2–11
- definedPartitions function 2–12
- disconnectService procedure 2–12
- findServiceRecord function 2–12
- getASConnectString function 2–13
- getConnectionParams function 2–13
- getConnectionString function 2–13
- getJMSPartitions function 2–14
- getJMSPtnInfo function 2–14
- getPartitionsByType function 2–14
- getPhysicalService function 2–15
- getServiceField function 2–15
- getServiceHandle function 2–15
- getServiceList function 2–16
- initializeSelf procedure 2–16
- isConnected function 2–16
- isDefaultService function 2–17
- loadPartitionInfo procedure 2–17
- parseConnectionParams function 2–17
- plipShutdown procedure 2–17
- reconnectService procedure 2–18
- registerService procedure 2–18
- savePartitionInfo procedure 2–18
- security_prompt procedure 2–19

setServiceHandle function 2–19

appServerConnect procedure
 AppServer Connection Manager 2–11

appServerDisconnect procedure
 AppServer Connection Manager 2–11

areFieldsCached function
 Security Manager 9–6

areTokensCached function
 Security Manager 9–6

askQuestion procedure
 Session Manager 10–7

authenticateUser procedure
 Security Manager 9–7

B

buildClientCache procedure
 Localization Manager 5–4
 Profile Manager 6–4

buildWidgetTable procedure
 Localization Manager 5–4

C

cacheEntity procedure

- General Manager 4–7
- cacheEntityMapping procedure
 - General Manager 4–7
- cacheGlobalSecurityAllocations procedure
 - Security Manager 9–7
- cacheGlobalSecurityStructures procedure
 - Security Manager 9–7
- calculateObjectPaths procedure
 - Repository Manager 8–12
- changeObjectInstance procedure
 - Repository Design Manager 8–24
- changeObjectType procedure
 - Repository Design Manager 8–25
- changePassword procedure
 - Security Manager 9–8
- checkIfOverlaps procedure
 - General Manager 4–8
- checkProfileDataExists procedure
 - Profile Manager 6–4
- checkUser procedure
 - Security Manager 9–9
- classHasAttribute function
 - Repository Design Manager 8–26
 - Repository Manager 8–14
- classIsA function
 - Repository Manager 8–14
- clearClientCache procedure
 - Localization Manager 5–5
 - Profile Manager 6–5
 - Repository Manager 8–14
 - Security Manager 9–10
- clearDesignCache procedure
 - Repository Design Manager 8–26
- Configuration File Manager
 - detectFileType function 1–5
 - expandTokens function 1–5
 - findFile function 1–6
 - getCodePath function 1–6
 - getExpandablePropertyValue function 1–6
 - getManagerHandle function 1–7
 - getPhysicalSessionType function 1–7
 - getProcedureHandle function 1–7
 - getSessionParam function 1–8
 - initializeSession procedure 1–8
 - isConfigManRunning function 1–8
 - isICFRunning function 1–9
 - obtainCFMTables procedure 1–9
 - obtainRegistryKeys procedure 1–9
 - parseConfig procedure 1–10
 - propertyExpander procedure 1–10
 - sessionShutdown procedure 1–10
 - setICFIsRunning function 1–11
 - setSessionParam function 1–11
 - subscribeAll procedure 1–11
- Connection Manager
 - connectService procedure 2–5
 - disconnectService procedure 2–5
 - getConnectionParams function 2–5
 - getConnectionString function 2–6
 - getPhysicalService function 2–6
 - getServiceHandle function 2–6
 - getServiceList function 2–7
 - getServiceSTManager function 2–7
 - getServiceType function 2–7
 - getServiceTypeManager function 2–8
 - initializeServices procedure 2–8
 - isConnected function 2–8
 - obtainConnectionTables procedure 2–9
 - plipShutdown procedure 2–9
 - reconnectService procedure 2–9
 - registerService procedure 2–10
 - setServiceTypeManager function 2–10
- connectService procedure
 - AppServer Connection Manager 2–11
 - Connection Manager 2–5
 - Database Connection Manager 2–20
- contextHelp procedure
 - Session Manager 10–9
- convertTimeToInteger function
 - General Manager 4–9

copyObjectMaster procedure
Repository Design Manager 8–27

createClassCache procedure
Repository Manager 8–15

createFolder function
General Manager 4–9

createGroupAllocation procedure
Security Manager 9–11

createGroupFromUser procedure
Security Manager 9–11

createLinks procedure
Session Manager 10–9

Customization Manager
getClientResultCodes function 3–3
getCustomisationTypesPrioritised
function 3–3
getReferenceLanguage function 3–3
getReferenceLoginCompany function
3–4
getReferenceSystem function 3–4
getReferenceUIType function 3–4
getReferenceUser function 3–4
getReferenceUserCategory function 3–5
getSessionCustomisationReferences
function 3–5
getSessionResultCodes function 3–5
InitializeObject procedure 3–5
PlipShutdown procedure 3–6
setClientResultCodes procedure 3–6
setSessionResultCodes procedure 3–7

D

Database Connection Manager
connectService procedure 2–20
disconnectService procedure 2–20
findServiceRecord function 2–20
getConnectionParams function 2–21
getConnectionString function 2–21
getPhysicalService function 2–21
getServiceField function 2–22
getServiceHandle function 2–22
getServiceList function 2–22

isConnected function 2–23
isDefaultService function 2–23
parseConnectionParams function 2–23
plipShutdown procedure 2–24
registerService procedure 2–24
setServiceHandle function 2–24

definedPartitions function
AppServer Connection Manager 2–12

deletePersistentProc procedure
Session Manager 10–10

deleteSessionProfile procedure
Profile Manager 6–6

destroyClassCache procedure
Repository Manager 8–15

detectFileType function
Configuration File Manager 1–5

disconnectService procedure
AppServer Connection Manager 2–12
Connection Manager 2–5
Database Connection Manager 2–20

E

escapeData function
UI Manager 11–4

expandTokens function
Configuration File Manager 1–5

extractRootFile procedure
Repository Manager 8–16

F

fieldandtokenSecurityCheck procedure
Security Manager 9–12

fieldSecurityCheck procedure
Security Manager 9–13

fieldSecurityGet procedure
Security Manager 9–13

findFile function

Configuration File Manager 1–6

findServiceRecord function

AppServer Connection Manager 2–12

Database Connection Manager 2–20

fixQueryString function

Session Manager 10–10

formatPersonDetails function

General Manager 4–10

G

General Manager

cacheEntity procedure 4–7

cacheEntityMapping procedure 4–7

checkIfOverlaps procedure 4–8

convertTimeToInteger function 4–9

createFolder function 4–9

formatPersonDetails function 4–10

getDBsForImportedEntities procedure
4–10

getDBVersion procedure 4–11

getDumpName procedure 4–11

getEntityCacheBuffer function 4–12

getEntityDescription procedure 4–12

getEntityDetail procedure 4–13

getEntityDisplayField procedure 4–13

getEntityExists procedure 4–14

getEntityFieldCacheBuffer function
4–14

getEntityTableName procedure 4–15

getHighKey function 4–15

getInternalEntries function 4–16

getKeyField function 4–16

getLanguageText procedure 4–16

getNextSequenceValue function 4–18

getObjField function 4–18

getOEMCode function 4–19

getOEMDescription function 4–19

getPropertyFromList function 4–19

getRecordCheckAudit procedure 4–20

getRecordCheckComment procedure
4–20

getRecordDetail procedure 4–21

getRecordUserProp procedure 4–21

getSequenceConfirmation function 4–22

getSequenceExist procedure 4–22

getSequenceMask procedure 4–22

getSiteNumber procedure 4–23

getStatusObj function 4–23

getStatusShortDesc function 4–24

getStatusTLA function 4–24

getTableDumpName function 4–24

getTableInfo procedure 4–25

getTableInfoObj procedure 4–25

getUpdatableTableInfo function 4–26

getUpdatableTableInfoObj function
4–26

getUserSourceLanguage procedure 4–27

haveOutstandingUpdates function 4–27

listLookup function 4–27

plipShutdown procedure 4–28

refreshMnemonicsCache procedure 4–28

setPropertyInList function 4–28

setWidgetAttribute function 4–29

updateTableViaSDO procedure 4–29

validateEntityMnemonic procedure 4–30

generateCalculatedField procedure

Repository Design Manager 8–28

generateClassCache procedure

Repository Design Manager 8–28

generateDataFields procedure

Repository Design Manager 8–29

generateDataLogicObject procedure

Repository Design Manager 8–30

generateDataObject procedure

Repository Design Manager 8–31

generateDynamicBrowse procedure

Repository Design Manager 8–32

generateDynamicSDF procedure

Repository Design Manager 8–33

generateDynamicViewer procedure

Repository Design Manager 8–34

generateEntityInstances procedure

Repository Design Manager 8–35

generateEntityObject procedure

- Repository Design Manager 8–36
- generateSBODataLogicObject procedure
 - Repository Design Manager 8–37
- generateSDOInstances procedure
 - Repository Design Manager 8–38
- generateVisualObject procedure
 - Repository Design Manager 8–39
- getActionUnderway procedure
 - Session Manager 10–10
- getASConnectString function
 - AppServer Connection Manager 2–13
- getBufferDbName function
 - Repository Design Manager 8–40
- getCacheClassBuffer function
 - Repository Manager 8–16
- getCacheLinkBuffer function
 - Repository Manager 8–17
- getCacheObjectBuffer function
 - Repository Manager 8–17
- getCachePageBuffer function
 - Repository Manager 8–17
- getClassChildren function
 - Repository Manager 8–18
- getClassFromInstance function
 - Repository Manager 8–18
- getClientCacheDir procedure
 - Repository Manager 8–18
- getClientResultCodes function
 - Customization Manager 3–3
- getCodePath function
 - Configuration File Manager 1–6
- getConnectionParams function
 - AppServer Connection Manager 2–13
 - Connection Manager 2–5
 - Database Connection Manager 2–21
- getConnectionString function
 - AppServer Connection Manager 2–13
 - Connection Manager 2–6
 - Database Connection Manager 2–21
- getContainerIcons procedure
 - Security Manager 9–14
- getCurrentLogicalName function
 - Repository Manager 8–19
- getCurrentLogicalName procedure
 - Session Manager 10–11
- getCustomisationTypesPrioritised function
 - Customization Manager 3–3
- getDBsForImportedEntities procedure
 - General Manager 4–10
- getDBVersion procedure
 - General Manager 4–11
- getDumpName procedure
 - General Manager 4–11
- getEntityCacheBuffer function
 - General Manager 4–12
- getEntityDescription procedure
 - General Manager 4–12
- getEntityDetail procedure
 - General Manager 4–13
- getEntityDisplayField procedure
 - General Manager 4–13
- getEntityExists procedure
 - General Manager 4–14
- getEntityFieldCacheBuffer function
 - General Manager 4–14
- getEntityTableName procedure
 - General Manager 4–15
- getExpandablePropertyValue function
 - Configuration File Manager 1–6
- getFieldSecurity procedure
 - Security Manager 9–14

getGlobalControl procedure Session Manager 10–11	General Manager 4–19
getHelp procedure Session Manager 10–12	getOEMDescription function General Manager 4–19
getHighKey function General Manager 4–15	getPartitionsByType function AppServer Connection Manager 2–14
getInternalEntries function General Manager 4–16	getPersistentProcs procedure Session Manager 10–13
getInternalEntryExists function Session Manager 10–12	getPhysicalService function AppServer Connection Manager 2–15 Connection Manager 2–6 Database Connection Manager 2–21
getJMSPartitions function AppServer Connection Manager 2–14	getPhysicalSessionType function Configuration File Manager 1–7
getJMSPtnInfo function AppServer Connection Manager 2–14	getProcedureHandle function Configuration File Manager 1–7
getKeyField function General Manager 4–16	getProductModuleList function Repository Design Manager 8–40
getLanguageText procedure General Manager 4–16	getProfileData procedure Profile Manager 6–6
getLoginUserInfo procedure Session Manager 10–12	getProfileTTHandle function Profile Manager 6–7
getManagerHandle function Configuration File Manager 1–7	getPropertyFromList function General Manager 4–19
getMandatoryTables procedure Security Manager 9–15	getPropertyList function Session Manager 10–13
getMappedFilename function Repository Manager 8–19	getRecordCheckAudit procedure General Manager 4–20
getNextSequenceValue function General Manager 4–18	getRecordCheckComment procedure General Manager 4–20
getObjectNames procedure Repository Manager 8–20	getRecordDetail procedure General Manager 4–21
getObjectSuperProcedure procedure Repository Manager 8–20	getRecordUserProp procedure General Manager 4–21
getObjField function General Manager 4–18	getReferenceLanguage function Customization Manager 3–3
getOEMCode function	

- getReferenceLoginCompany function
 - Customization Manager 3–4
- getReferenceSystem function
 - Customization Manager 3–4
- getReferenceUIType function
 - Customization Manager 3–4
- getReferenceUser function
 - Customization Manager 3–4
- getReferenceUserCategory function
 - Customization Manager 3–5
- getSchemaQueryHandle function
 - Repository Design Manager 8–41
- getSecurityControl procedure
 - Security Manager 9–16
- getSequenceConfirmation function
 - General Manager 4–22
- getSequenceExist procedure
 - General Manager 4–22
- getSequenceMask procedure
 - General Manager 4–22
- getServiceField function
 - AppServer Connection Manager 2–15
 - Database Connection Manager 2–22
- getServiceHandle function
 - AppServer Connection Manager 2–15
 - Connection Manager 2–6
 - Database Connection Manager 2–22
- getServiceList function
 - AppServer Connection Manager 2–16
 - Connection Manager 2–7
 - Database Connection Manager 2–22
- getServiceSTManager function
 - Connection Manager 2–7
- getServiceType function
 - Connection Manager 2–7
- getServiceTypeManager function
 - Connection Manager 2–8
- getSessionCustomisationReferences function
 - Customization Manager 3–5
- getSessionParam function
 - Configuration File Manager 1–8
- getSessionResultCodes function
 - Customization Manager 3–5
- getSiteNumber procedure
 - General Manager 4–23
- getStatusObj function
 - General Manager 4–23
- getStatusShortDesc function
 - General Manager 4–24
- getStatusTLA function
 - General Manager 4–24
- getTableDumpName function
 - General Manager 4–24
- getTableInfo procedure
 - General Manager 4–25
- getTableInfoObj procedure
 - General Manager 4–25
- getToolbarBandActions procedure
 - Repository Manager 8–21
- getTranslation procedure
 - Localization Manager 5–6
- getUpdatableTableInfo function
 - General Manager 4–26
- getUpdatableTableInfoObj function
 - General Manager 4–26
- getUserSourceLanguage procedure
 - General Manager 4–27
- getWidgetSizeFromFormat function
 - Repository Design Manager 8–42

H

haveOutstandingUpdates function
General Manager 4–27

helpAbout procedure
Session Manager 10–14

helpContents procedure
Session Manager 10–14

helpHelp procedure
Session Manager 10–14

helpTopics procedure
Session Manager 10–15

htmlHelpKeywords procedure
Session Manager 10–15

htmlHelpTopic procedure
Session Manager 10–16

I

increaseFrameforPopup procedure
Session Manager 10–16

InitializeObject procedure
Customization Manager 3–5

initializeSelf procedure
AppServer Connection Manager 2–16

initializeServices procedure
Connection Manager 2–8

initializeSession procedure
Configuration File Manager 1–8

insertObjectInstance procedure
Repository Design Manager 8–42

insertObjectLinks procedure
Repository Design Manager 8–43

insertObjectMaster procedure
Repository Design Manager 8–44

insertObjectPage procedure

Repository Design Manager 8–46

insertUiEvents procedure
Repository Design Manager 8–47

IsA function
Repository Manager 8–22

isConfigManRunning function
Configuration File Manager 1–8

isConnected function
AppServer Connection Manager 2–16
Connection Manager 2–8
Database Connection Manager 2–23

isDefaultService function
AppServer Connection Manager 2–17
Database Connection Manager 2–23

isICFRunning function
Configuration File Manager 1–9

isObjQuoted function
Session Manager 10–16

K

killPlips procedure
Session Manager 10–17

killProcedure procedure
Session Manager 10–18

L

launchContainer procedure
Session Manager 10–18

LaunchExternalProcess procedure
Session Manager 10–20

launchProcedure procedure
Session Manager 10–20

listLookup function
General Manager 4–27

loadPartitionInfo procedure

AppServer Connection Manager 2–17

Localization Manager

- buildClientCache procedure 5–4
- buildWidgetTable procedure 5–4
- clearClientCache procedure 5–5
- getTranslation procedure 5–6
- multiTranslation procedure 5–7
- plipShutdown procedure 5–8
- receiveCacheClient procedure 5–8
- translatePhrase function 5–8
- translateWidgetTable procedure 5–9
- updateTranslations procedure 5–9

M

- menuItemSecurityCheck procedure
 - Security Manager 9–16
- menuStructureSecurityCheck procedure
 - Security Manager 9–17
- multiTranslation procedure
 - Localization Manager 5–7

N

- notifyUser procedure
 - Session Manager 10–22

O

- ObjectExists function
 - Repository Design Manager 8–47
- objectSecurityCheck procedure
 - Security Manager 9–17
- obtainCFMTables procedure
 - Configuration File Manager 1–9
- obtainConnectionTables procedure
 - Connection Manager 2–9
- obtainRegistryKeys procedure
 - Configuration File Manager 1–9

P

- parseConfig procedure
 - Configuration File Manager 1–10
- parseConnectionParams function
 - AppServer Connection Manager 2–17
 - Database Connection Manager 2–23
- PlipShutdown procedure
 - Customization Manager 3–6
- plipShutdown procedure
 - AppServer Connection Manager 2–17
 - Connection Manager 2–9
 - Database Connection Manager 2–24
 - General Manager 4–28
 - Localization Manager 5–8
 - Profile Manager 6–7
 - Repository Design Manager 8–47
 - Repository Manager 8–22
 - Security Manager 9–18
 - Session Manager 10–22
- prepareObjectName function
 - Repository Design Manager 8–48
- processRequest procedure
 - Web Request Manager 12–3
- Profile Manager
 - buildClientCache procedure 6–4
 - checkProfileDataExists procedure 6–4
 - clearClientCache procedure 6–5
 - deleteSessionProfile procedure 6–6
 - getProfileData procedure 6–6
 - getProfileTTHandle function 6–7
 - plipShutdown procedure 6–7
 - receiveProfileCache procedure 6–8
 - setProfileData procedure 6–8
 - updateCacheToDb procedure 6–10
- propertyExpander procedure
 - Configuration File Manager 1–10

R

- rangeSecurityCheck procedure
 - Security Manager 9–18

- receiveCacheClient procedure
 - Localization Manager 5–8
- receiveCacheSessionSecurity procedure
 - Security Manager 9–19
- receiveProfileCache procedure
 - Profile Manager 6–8
- reconnectService procedure
 - AppServer Connection Manager 2–18
 - Connection Manager 2–9
- Referential Integrity Manager
 - versionData procedure 7–5
- refreshMnemonicsCache procedure
 - General Manager 4–28
- registerService procedure
 - AppServer Connection Manager 2–18
 - Connection Manager 2–10
 - Database Connection Manager 2–24
- removeAttributeValues procedure
 - Repository Design Manager 8–49
- removeObject procedure
 - Repository Design Manager 8–50
- removeObjectInstance procedure
 - Repository Design Manager 8–51
- removeObjectPage procedure
 - Repository Design Manager 8–52
- removePageInstance procedure
 - Repository Design Manager 8–53
- removeUIEvents procedure
 - Repository Design Manager 8–54
- Repository Design Manager
 - changeObjectInstance procedure 8–24
 - changeObjectType procedure 8–25
 - classHasAttribute function 8–26
 - clearDesignCache procedure 8–26
 - copyObjectMaster procedure 8–27
 - generateCalculatedField procedure 8–28
 - generateClassCache procedure 8–28
 - generateDataFields procedure 8–29
 - generateDataLogicObject procedure 8–30
 - generateDataObject procedure 8–31
 - generateDynamicBrowse procedure 8–32
 - generateDynamicSDF procedure 8–33
 - generateDynamicViewer procedure 8–34
 - generateEntityInstances procedure 8–35
 - generateEntityObject procedure 8–36
 - generateSBODDataLogicObject procedure 8–37
 - generateSDOInstances procedure 8–38
 - generateVisualObject procedure 8–39
 - getBufferDbName function 8–40
 - getProductModuleList function 8–40
 - getSchemaQueryHandle function 8–41
 - getWidgetSizeFromFormat function 8–42
 - insertObjectInstance procedure 8–42
 - insertObjectLinks procedure 8–43
 - insertObjectMaster procedure 8–44
 - insertObjectPage procedure 8–46
 - insertUiEvents procedure 8–47
 - ObjectExists function 8–47
 - plipShutdown procedure 8–47
 - prepareObjectName function 8–48
 - removeAttributeValues procedure 8–49
 - removeObject procedure 8–50
 - removeObjectInstance procedure 8–51
 - removeObjectPage procedure 8–52
 - removePageInstance procedure 8–53
 - removeUIEvents procedure 8–54
 - setQualifiedTableName function 8–54
- Repository Manager
 - calculateObjectPaths procedure 8–12
 - classHasAttribute function 8–14
 - classIsA function 8–14
 - clearClientCache procedure 8–14
 - createClassCache procedure 8–15
 - destroyClassCache procedure 8–15
 - extractRootFile procedure 8–16
 - getCacheClassBuffer function 8–16
 - getCacheLinkBuffer function 8–17
 - getCacheObjectBuffer function 8–17
 - getCachePageBuffer function 8–17
 - getClassChildren function 8–18
 - getClassFromInstance function 8–18
 - getClientCacheDir procedure 8–18

- getCurrentLogicalName function 8–19
 - getMappedFilename function 8–19
 - getObjectNames procedure 8–20
 - getObjectSuperProcedure procedure 8–20
 - getToolBarBandActions procedure 8–21
 - IsA function 8–22
 - plipShutdown procedure 8–22
 - resolveResultCodes procedure 8–22
 - startDataObject procedure 8–23
 - storeAttributeValues procedure 8–23
- resizeLookupFrame procedure
 - Session Manager 10–23
- resizeNormalFrame procedure
 - Session Manager 10–23
- resizeSDFFrame procedure
 - Session Manager 10–24
- resolveResultCodes procedure
 - Repository Manager 8–22
- runLookup procedure
 - Session Manager 10–24
- S**
- savePartitionInfo procedure
 - AppServer Connection Manager 2–18
- Security Manager
 - areFieldsCached function 9–6
 - areTokensCached function 9–6
 - authenticateUser procedure 9–7
 - cacheGlobalSecurityAllocations procedure 9–7
 - cacheGlobalSecurityStructures procedure 9–7
 - changePassword procedure 9–8
 - checkUser procedure 9–9
 - clearClientCache procedure 9–10
 - createGroupAllocation procedure 9–11
 - createGroupFromUser procedure 9–11
 - fieldandtokenSecurityCheck procedure 9–12
 - fieldSecurityCheck procedure 9–13
 - fieldSecurityGet procedure 9–13
 - getContainerIcons procedure 9–14
 - getFieldSecurity procedure 9–14
 - getMandatoryTables procedure 9–15
 - getSecurityControl procedure 9–16
 - menuItemSecurityCheck procedure 9–16
 - menuStructureSecurityCheck procedure 9–17
 - objectSecurityCheck procedure 9–17
 - plipShutdown procedure 9–18
 - rangeSecurityCheck procedure 9–18
 - receiveCacheSessionSecurity procedure 9–19
 - tableSecurityCheck procedure 9–19
 - tokenSecurityCheck procedure 9–20
 - tokenSecurityGet procedure 9–20
 - updateUserAllocations procedure 9–21
 - userLoginOrganisations procedure 9–22
 - userSecurityCheck procedure 9–22
- security_prompt procedure
 - AppServer Connection Manager 2–19
- sendEmail procedure
 - Session Manager 10–25
- Session Manager
 - activateSession procedure 10–7
 - askQuestion procedure 10–7
 - contextHelp procedure 10–9
 - createLinks procedure 10–9
 - deletePersistentProc procedure 10–10
 - fixQueryString function 10–10
 - getActionUnderway procedure 10–10
 - getCurrentLogicalName procedure 10–11
 - getGlobalControl procedure 10–11
 - getHelp procedure 10–12
 - getInternalEntryExists function 10–12
 - getLoginUserInfo procedure 10–12
 - getPersistentProcs procedure 10–13
 - getPropertyList function 10–13
 - helpAbout procedure 10–14
 - helpContents procedure 10–14
 - helpHelp procedure 10–14
 - helpTopics procedure 10–15
 - htmlHelpKeywords procedure 10–15
 - htmlHelpTopic procedure 10–16
 - increaseFrameforPopup procedure 10–16
 - isObjQuoted function 10–16

- killPlips procedure 10–17
- killProcedure procedure 10–18
- launchContainer procedure 10–18
- LaunchExternalProcess procedure 10–20
- launchProcedure procedure 10–20
- notifyUser procedure 10–22
- plipShutdown procedure 10–22
- resizeLookupFrame procedure 10–23
- resizeNormalFrame procedure 10–23
- resizeSDFFrame procedure 10–24
- runLookup procedure 10–24
- sendEmail procedure 10–25
- setActionUnderway procedure 10–26
- setAttributesInObject procedure 10–27
- setPropertyList function 10–27
- setReturnValue procedure 10–28
- setSecurityForDynObjects function 10–28
- showMessages procedure 10–29
- showWarningMessages procedure 10–30
- translateWidgets procedure 10–31
- updateErrorLog procedure 10–31
- updateHelp procedure 10–32
- widgetWalk procedure 10–32
- sessionShutdown procedure
 - Configuration File Manager 1–10
- setActionUnderway procedure
 - Session Manager 10–26
- setAttributesInObject procedure
 - Session Manager 10–27
- setClientAction procedure
 - UI Manager 11–4
- setClientResultCodes procedure
 - Customization Manager 3–6
- setICFIsRunning function
 - Configuration File Manager 1–11
- setProfileData procedure
 - Profile Manager 6–8
- setPropertyList function
 - Session Manager 10–27
- setPropertyValueInList function
 - General Manager 4–28
- setQualifiedTableName function
 - Repository Design Manager 8–54
- setReturnValue procedure
 - Session Manager 10–28
- setSecurityForDynObjects function
 - Session Manager 10–28
- setServiceHandle function
 - AppServer Connection Manager 2–19
 - Database Connection Manager 2–24
- setServiceTypeManager function
 - Connection Manager 2–10
- setSessionParam function
 - Configuration File Manager 1–11
- setSessionResultCodes procedure
 - Customization Manager 3–7
- setWidgetAttribute function
 - General Manager 4–29
- showMessages procedure
 - Session Manager 10–29
- showWarningMessages procedure
 - Session Manager 10–30
- startDataObject procedure
 - Repository Manager 8–23
- storeAttributeValues procedure
 - Repository Manager 8–23
- subscribeAll procedure
 - Configuration File Manager 1–11

T

- tableSecurityCheck procedure
 - Security Manager 9–19
- tokenSecurityCheck procedure
 - Security Manager 9–20
- tokenSecurityGet procedure

Security Manager 9–20

translatePhrase function

Localization Manager 5–8

translateWidgets procedure

Session Manager 10–31

translateWidgetTable procedure

Localization Manager 5–9

U

updateCacheToDb procedure

Profile Manager 6–10

updateErrorLog procedure

Session Manager 10–31

updateHelp procedure

Session Manager 10–32

updateTableViaSDO procedure

General Manager 4–29

updateTranslations procedure

Localization Manager 5–9

updateUserAllocations procedure

Security Manager 9–21

User Interface Manager

escapeData function 11–4

setClientAction procedure 11–4

userLoginOrganisations procedure

Security Manager 9–22

userSecurityCheck procedure

Security Manager 9–22

V

validateEntityMnemonic procedure

General Manager 4–30

versionData procedure

Referential Integrity Manager 7–5

W

Web Request Manager

processRequest procedure 12–3

widgetWalk procedure

Session Manager 10–32

