## Contents

**Preface** ................................................................. Preface–1

1. **Configuration File Manager** .................................. 1–1
   - Overview ................................................................. 1–2
   - Temp-tables used by Configuration File Manager ............ 1–3
   - Configuration File Manager APIs 
     - detectFileType ......................................................... 1–4
     - expandTokens ......................................................... 1–4
     - findFile ................................................................. 1–5
     - getCodePath .......................................................... 1–5
     - getExpandablePropertyValue ..................................... 1–6
     - getManagerHandle .................................................. 1–6
     - getPhysicalSessionType ......................................... 1–6
     - getProcedureHandle ................................................ 1–7
     - getSessionParam .................................................... 1–7
     - initializeSession ................................................... 1–7
     - isConfigManRunning ............................................... 1–8
     - isICFIsRunning ..................................................... 1–8
     - obtainCFMTables .................................................... 1–8
     - obtainRegistryKeys ............................................... 1–9
     - parseConfig .......................................................... 1–9
     - propertyExpander ................................................... 1–9
     - sessionShutdown .................................................... 1–10
     - setICFIsRunning .................................................... 1–10
     - setSessionParam ................................................... 1–10
     - subscribeAll ........................................................ 1–11

2. **Connection and Service Type Managers** ...................... 2–1
   - Overview ................................................................. 2–2
   - Temp-tables used by Connection and Service Type Managers 2–4
   - Connection Manager APIs 
     - connectService ..................................................... 2–5
     - connectServiceWithParams ....................................... 2–5
     - disconnectService ................................................ 2–6
     - getConnectionParams ............................................... 2–6
<table>
<thead>
<tr>
<th>Database Connection Manager APIs</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getConnectionString</td>
<td>2–6</td>
</tr>
<tr>
<td>getPhysicalService</td>
<td>2–6</td>
</tr>
<tr>
<td>getServiceHandle</td>
<td>2–7</td>
</tr>
<tr>
<td>getServiceList</td>
<td>2–7</td>
</tr>
<tr>
<td>getServiceSTManager</td>
<td>2–7</td>
</tr>
<tr>
<td>getServiceType</td>
<td>2–7</td>
</tr>
<tr>
<td>initializeServices</td>
<td>2–8</td>
</tr>
<tr>
<td>isDefaultService</td>
<td>2–8</td>
</tr>
<tr>
<td>isConnected</td>
<td>2–8</td>
</tr>
<tr>
<td>isServiceRegistered</td>
<td>2–9</td>
</tr>
<tr>
<td>obtainConnectionTables</td>
<td>2–9</td>
</tr>
<tr>
<td>plipShutdown</td>
<td>2–9</td>
</tr>
<tr>
<td>reconnectService</td>
<td>2–10</td>
</tr>
<tr>
<td>registerService</td>
<td>2–10</td>
</tr>
<tr>
<td>setServiceTypeManager</td>
<td>2–10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AppServer Connection Manager APIs</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>appServerConnect</td>
<td>2–11</td>
</tr>
<tr>
<td>appServerDisconnect</td>
<td>2–11</td>
</tr>
<tr>
<td>connectService</td>
<td>2–11</td>
</tr>
<tr>
<td>connectServiceWithParams</td>
<td>2–11</td>
</tr>
<tr>
<td>definedPartitions</td>
<td>2–12</td>
</tr>
<tr>
<td>disconnectService</td>
<td>2–12</td>
</tr>
<tr>
<td>findServiceRecord</td>
<td>2–12</td>
</tr>
<tr>
<td>getASConnectString</td>
<td>2–13</td>
</tr>
<tr>
<td>getConnectionParams</td>
<td>2–13</td>
</tr>
<tr>
<td>getConnectionString</td>
<td>2–14</td>
</tr>
<tr>
<td>getJMSPartitions</td>
<td>2–14</td>
</tr>
<tr>
<td>getJMSPtlnfo</td>
<td>2–14</td>
</tr>
<tr>
<td>getPartitionsByType</td>
<td>2–14</td>
</tr>
<tr>
<td>getPhysicalService</td>
<td>2–14</td>
</tr>
<tr>
<td>getServiceField</td>
<td>2–15</td>
</tr>
<tr>
<td>getServiceHandle</td>
<td>2–15</td>
</tr>
<tr>
<td>getServiceList</td>
<td>2–16</td>
</tr>
<tr>
<td>initializeSelf</td>
<td>2–16</td>
</tr>
<tr>
<td>isDefaultService</td>
<td>2–16</td>
</tr>
<tr>
<td>loadPartitionInfo</td>
<td>2–17</td>
</tr>
<tr>
<td>parseConnectionParams</td>
<td>2–17</td>
</tr>
<tr>
<td>plipShutdown</td>
<td>2–17</td>
</tr>
<tr>
<td>reconnectService</td>
<td>2–17</td>
</tr>
<tr>
<td>registerService</td>
<td>2–18</td>
</tr>
<tr>
<td>savePartitionInfo</td>
<td>2–18</td>
</tr>
<tr>
<td>security_prompt</td>
<td>2–18</td>
</tr>
<tr>
<td>setServiceHandle</td>
<td>2–18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Database Connection Manager APIs</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectService</td>
<td>2–19</td>
</tr>
<tr>
<td>connectServiceWithParams</td>
<td>2–19</td>
</tr>
<tr>
<td>disconnectService</td>
<td>2–19</td>
</tr>
<tr>
<td>findServiceRecord</td>
<td>2–20</td>
</tr>
<tr>
<td>getConnectionParams</td>
<td>2–20</td>
</tr>
<tr>
<td>getConnectionString</td>
<td>2–20</td>
</tr>
<tr>
<td>getPhysicalService</td>
<td>2–21</td>
</tr>
<tr>
<td>getServiceField</td>
<td>2–21</td>
</tr>
<tr>
<td>getServiceHandle</td>
<td>2–21</td>
</tr>
<tr>
<td>getServiceList</td>
<td>2–22</td>
</tr>
<tr>
<td>isConnected</td>
<td>2–22</td>
</tr>
<tr>
<td>isDefaultService</td>
<td>2–22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Connection Manager APIs</td>
<td>Page</td>
</tr>
<tr>
<td>connectService</td>
<td>2–19</td>
</tr>
<tr>
<td>connectServiceWithParams</td>
<td>2–19</td>
</tr>
<tr>
<td>disconnectService</td>
<td>2–19</td>
</tr>
<tr>
<td>findServiceRecord</td>
<td>2–20</td>
</tr>
<tr>
<td>getConnectionParams</td>
<td>2–20</td>
</tr>
<tr>
<td>getConnectionString</td>
<td>2–20</td>
</tr>
<tr>
<td>getPhysicalService</td>
<td>2–21</td>
</tr>
<tr>
<td>getServiceField</td>
<td>2–21</td>
</tr>
<tr>
<td>getServiceHandle</td>
<td>2–21</td>
</tr>
<tr>
<td>getServiceList</td>
<td>2–22</td>
</tr>
<tr>
<td>isConnected</td>
<td>2–22</td>
</tr>
<tr>
<td>isDefaultService</td>
<td>2–22</td>
</tr>
</tbody>
</table>
3. **Customization Manager** 3–1
   
   **Overview** 3–2
   
   **Customization Manager APIs** 3–3
   
   - getClientResultCodes 3–3
   - getCustomisationTypesPrioritised 3–3
   - getReferenceLanguage 3–3
   - getReferenceLoginCompany 3–4
   - getReferenceSystem 3–4
   - getReferenceUIType 3–4
   - getReferenceUser 3–4
   - getReferenceUserCategory 3–5
   - getSessionCustomisationReferences 3–5
   - getSessionResultCodes 3–5
   - InitializeObject 3–5
   - plipShutdown 3–6
   - setClientResultCodes 3–6
   - setSessionResultCodes 3–6

4. **General Manager** 4–1
   
   **Overview** 4–2
   
   **Temp-tables used by General Manager** 4–3
   
   **General Manager APIs** 4–5
   
   - checkIfOverlaps 4–5
   - convertTimeToInteger 4–6
   - createFolder 4–7
   - formatPersonDetails 4–7
   - getDBsForImportedEntities 4–7
   - getDBVersion 4–8
   - getDumpName 4–8
   - getEntityCacheBuffer 4–9
   - getEntityDescription 4–9
   - getEntityDetail 4–10
   - getEntityDisplayField 4–10
5. Localization Manager ................................. 5–1
   Overview ........................................... 5–2
   Temp-tables used by Localization Manager .... 5–3
   Localization Manager APIs ...................... 5–4
      buildClientCache ................................ 5–4
      buildWidgetTable ................................ 5–4
      clearClientCache ................................ 5–5
      getTranslation .................................. 5–5
      multiTranslation ................................ 5–6
      plipShutdown .................................... 5–7
      receiveCacheClient .............................. 5–7
      translatePhrase ................................. 5–7
      translateSingleObject ......................... 5–8
      translateWidgetTable ......................... 5–9
      updateTranslations ............................ 5–9

6. Profile Manager ...................................... 6–1
   Overview .......................................... 6–2
7. Referential Integrity Manager ........................................ 7–1
   Overview ......................................................... 7–2
   Temp-tables used by Referential Integrity Manager ........... 7–3
   Referential Integrity Manager API .............................. 7–4
      versionData .................................................... 7–4

8. Repository Managers ..................................................... 8–1
   Overview ......................................................... 8–2
   Temp-tables used by Repository Managers ...................... 8–4
      Temp-tables used by Repository Manager .................... 8–4
      Temp-tables used by Repository Design Manager .......... 8–7
   Repository Manager APIs ........................................ 8–11
      calculateObjectPaths ........................................ 8–11
      classHasAttribute .......................................... 8–12
      classIsA ....................................................... 8–13
      clearClientCache ............................................ 8–13
      createClassCache ........................................... 8–13
      destroyClassCache ........................................... 8–14
      extractRootFile .............................................. 8–14
      getCacheClassBuffer ........................................ 8–14
      getCacheLinkBuffer ......................................... 8–15
      getCacheObjectBuffer ....................................... 8–15
      getCachePageBuffer ......................................... 8–15
      getClassChildren ............................................ 8–16
      getClassFromInstance ....................................... 8–16
      getClientCacheDir .......................................... 8–16
      getCurrentLogicalName ...................................... 8–17
      getMappedFilename ......................................... 8–17
      getObjectNames ............................................. 8–18
      getObjectSuperProcedure ................................... 8–18
      getToolbarBandActions ...................................... 8–19
      IsA ............................................................... 8–19
      plipShutdown .................................................. 8–20
      resolveResultCodes .......................................... 8–20
      saveEntitiesToClientCache ................................ 8–20
      startDataObject ............................................. 8–21
      storeAttributeValues ....................................... 8–21
   Repository Design Manager APIs ................................ 8–22
      changeObjectInstance ........................................ 8–22
      changeObjectType ............................................ 8–23
      classHasAttribute .......................................... 8–24
      clearDesignCache ............................................ 8–24
      copyObjectMaster ........................................... 8–25
generateCalculatedField .......................................................... 8–26
generateClassCache .............................................................. 8–26
generateDataFields ............................................................... 8–27
generateDataLogicObject ....................................................... 8–28
generateDataObject .............................................................. 8–29
generateDynamicBrowse ....................................................... 8–30
generateDynamicSDF ............................................................. 8–31
generateDynamicViewer ......................................................... 8–32
generateEntityInstances ......................................................... 8–33
generateEntityObject ............................................................ 8–34
generateSBODataLogicObject .................................................. 8–35
generateSDOInstances ............................................................ 8–36
generateVisualObject ............................................................ 8–37
getBufferDbName ................................................................. 8–38
getProductModuleList ........................................................... 8–38
getSchemaQueryHandle .......................................................... 8–39
getWidgetSizeFromFormat ....................................................... 8–39
insertClass .......................................................................... 8–40
insertObjectInstance ............................................................ 8–41
insertObjectLinks ................................................................. 8–42
insertObjectMaster ............................................................... 8–42
insertObjectPage ................................................................. 8–44
insertSupportedLink ............................................................. 8–44
insertUIEvents ................................................................. 8–46
ObjectExists ................................................................. 8–46
plopShutdown ................................................................. 8–46
prepareObjectName ............................................................. 8–47
removeAttributeValues ........................................................ 8–48
removeObject ................................................................. 8–48
removeObjectPage ............................................................. 8–49
removePageInstance ............................................................ 8–50
removeSupportedLink .......................................................... 8–50
removeUIEvents ............................................................... 8–51
setQualifiedTableName ......................................................... 8–51

9. Security Manager .......................................................... 9–1
   Overview ................................................................. 9–1
   Temp-Tables used by Security Manager ................................... 9–2
   Security Manager APIs ................................................... 9–5
      areFieldsCached ........................................................ 9–5
      areTokensCached ....................................................... 9–5
      authenticateUser ....................................................... 9–6
      cacheGlobalSecurityAllocations .................................... 9–6
      cacheGlobalSecurityStructures ...................................... 9–6
      changePassword ....................................................... 9–7
      checkUser ............................................................. 9–8
      clearClientCache ..................................................... 9–9
      createGroupAllocation ............................................... 9–9
      createGroupFromUser ................................................. 9–10
      fieldandtokenSecurityCheck ....................................... 9–10
      fieldSecurityCheck .................................................. 9–11
      fieldSecurityGet .................................................... 9–11
      getContainerIcons ..................................................... 9–12
      getFieldSecurity ...................................................... 9–12
      getMandatoryTables .................................................. 9–13
      getSecurityControl .................................................. 9–13
10. **Session Manager** .................................................. 10–1
   Overview ........................................................... 10–2
   Temp-tables used by Session Manager .............. 10–4
   Session Manager APIs ........................................... 10–6
       activateSession ................................................. 10–6
       addAsSuperProcedure ....................................... 10–7
       askQuestion ................................................ 10–7
       contextHelp ............................................... 10–8
       createLinks ................................................. 10–9
       deletePersistentProc .................................... 10–9
       deleteSession ............................................. 10–9
       establishSession ......................................... 10–10
       getActionUnderway ...................................... 10–10
       getCurrentLogicalName .................................. 10–11
       getGlobalControl ....................................... 10–11
       getHelp .................................................. 10–12
       getInternalEntryExists ................................. 10–13
       getLoginUserInfo ....................................... 10–13
       getPersistentProcs .................................... 10–13
       getPropertyList .......................................... 10–14
       getSessionList .......................................... 10–14
       getSessionTypeOverrideInfo ........................... 10–15
       helpAbout ................................................. 10–15
       helpContents .............................................. 10–16
       helpHelp .................................................. 10–16
       helpTopics ................................................ 10–16
       htmlHelpTopic ............................................ 10–17
       htmlHelpKeywords ...................................... 10–17
       increaseFrameforPopup .................................. 10–18
       isObjQuoted ............................................... 10–19
       killPlips .................................................. 10–19
       killProcedure ............................................. 10–20
       launchContainer ......................................... 10–21
       launchExternalProcess ................................ 10–22
       launchProcedure ....................................... 10–23
       notifyUser ............................................... 10–24
       pipShutdown .............................................. 10–25
       resizeLookupFrame ...................................... 10–25
       resizeNormalFrame ..................................... 10–26
       resizeSDFFrame ......................................... 10–26
       runLookup ................................................ 10–27
       sendEmail ............................................... 10–27
       setActionUnderway ..................................... 10–28
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>setAttributesInObject</td>
<td>10–29</td>
</tr>
<tr>
<td>setPropertyList</td>
<td>10–29</td>
</tr>
<tr>
<td>setReturnValue</td>
<td>10–30</td>
</tr>
<tr>
<td>setSecurityForDynObjects</td>
<td>10–30</td>
</tr>
<tr>
<td>setSessTypeOverrideInfo</td>
<td>10–30</td>
</tr>
<tr>
<td>showMessages</td>
<td>10–31</td>
</tr>
<tr>
<td>showWarningMessages</td>
<td>10–32</td>
</tr>
<tr>
<td>translateWidgets</td>
<td>10–33</td>
</tr>
<tr>
<td>updateErrorLog</td>
<td>10–33</td>
</tr>
<tr>
<td>updateHelp</td>
<td>10–33</td>
</tr>
<tr>
<td>widgetWalk</td>
<td>10–34</td>
</tr>
</tbody>
</table>

11. **User Interface Manager** ................................................. 11–1
   - Overview ......................................................... 11–2
     - User Interface Manager APIs .................................. 11–3
       - escapeData ............................................... 11–3
       - setClientAction ......................................... 11–3

12. **Web Request Manager** .................................................. 12–1
   - Overview ......................................................... 12–2
     - Web Request Manager API .................................... 12–2
       - processRequest ........................................... 12–2

**Index** .............................................................................. Index–1
## Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Configuration File Manager files</td>
<td>1-2</td>
</tr>
<tr>
<td>1-2</td>
<td>Temp-tables defined in af\sup2\afxmlcfgtt.i</td>
<td>1-3</td>
</tr>
<tr>
<td>1-3</td>
<td>Temp-table defined in af\sup2\afsvtnode.i</td>
<td>1-4</td>
</tr>
<tr>
<td>2-1</td>
<td>Connection Manager files</td>
<td>2-2</td>
</tr>
<tr>
<td>2-2</td>
<td>AppServer Connection Manager files</td>
<td>2-3</td>
</tr>
<tr>
<td>2-3</td>
<td>Database Connection Manager files</td>
<td>2-3</td>
</tr>
<tr>
<td>2-4</td>
<td>WebService Service Type Manager files</td>
<td>2-3</td>
</tr>
<tr>
<td>2-5</td>
<td>Temp-tables defined in af\app\afconntdef.i</td>
<td>2-4</td>
</tr>
<tr>
<td>2-6</td>
<td>Temp-tables defined in af\sup2\afsrvttype.i</td>
<td>2-4</td>
</tr>
<tr>
<td>3-1</td>
<td>Customization Manager files</td>
<td>3-2</td>
</tr>
<tr>
<td>4-1</td>
<td>General Manager files</td>
<td>4-2</td>
</tr>
<tr>
<td>4-2</td>
<td>Temp-table defined in af\app\afgenmngnp.i</td>
<td>4-3</td>
</tr>
<tr>
<td>4-3</td>
<td>Temp-table defined in af\app\gsts园区menn.i</td>
<td>4-4</td>
</tr>
<tr>
<td>4-4</td>
<td>Temp-table defined in af\app\gsmstttcch.i</td>
<td>4-5</td>
</tr>
<tr>
<td>5-1</td>
<td>Localization Manager files</td>
<td>5-2</td>
</tr>
<tr>
<td>5-2</td>
<td>Temp-tables defined in af\app\aftttranslation.i</td>
<td>5-3</td>
</tr>
<tr>
<td>5-3</td>
<td>Temp-tables defined in adm2\tttranslate.i</td>
<td>5-3</td>
</tr>
<tr>
<td>6-1</td>
<td>Profile Manager files</td>
<td>6-2</td>
</tr>
<tr>
<td>6-2</td>
<td>Temp-table defined in af\app\afttprofiledata.i</td>
<td>6-3</td>
</tr>
<tr>
<td>7-1</td>
<td>Referential Integrity Manager files</td>
<td>7-2</td>
</tr>
<tr>
<td>7-2</td>
<td>Temp-tables defined in ry\app\ryrisrvrp.p</td>
<td>7-3</td>
</tr>
<tr>
<td>8-1</td>
<td>Repository Manager files</td>
<td>8-3</td>
</tr>
<tr>
<td>8-2</td>
<td>Repository Design Manager files</td>
<td>8-3</td>
</tr>
<tr>
<td>8-3</td>
<td>Temp-tables defined in ry\inc\getobjecti.i</td>
<td>8-4</td>
</tr>
<tr>
<td>8-4</td>
<td>Temp-tables defined in ry\inc\ryrepatset.i</td>
<td>8-6</td>
</tr>
<tr>
<td>8-5</td>
<td>Temp-tables defined in ry\app\rymenufunc.i</td>
<td>8-6</td>
</tr>
<tr>
<td>8-6</td>
<td>Other temp-tables used by the Repository Manager</td>
<td>8-7</td>
</tr>
<tr>
<td>8-7</td>
<td>Temp-tables defined in ry\inc\ryreplnset.i</td>
<td>8-7</td>
</tr>
<tr>
<td>8-8</td>
<td>Temp-tables defined in ry\app\rygenomngnp.p</td>
<td>8-8</td>
</tr>
<tr>
<td>8-9</td>
<td>Temp-tables defined in ry\app\rydesmngrp.i</td>
<td>8-10</td>
</tr>
<tr>
<td>9-1</td>
<td>Security Manager files</td>
<td>9-2</td>
</tr>
<tr>
<td>9-2</td>
<td>Temp-tables defined in af\app\afsecmngnp.i</td>
<td>9-3</td>
</tr>
<tr>
<td>9-3</td>
<td>Temp-table defined in af\app\afsectttdef.i</td>
<td>9-4</td>
</tr>
<tr>
<td>9-4</td>
<td>Temp-table defined in af\app\afsectttctrl.i</td>
<td>9-4</td>
</tr>
<tr>
<td>10-1</td>
<td>Session Manager files</td>
<td>10-3</td>
</tr>
<tr>
<td>10-2</td>
<td>Temp-tables defined in af\app\afsesmngrp.i</td>
<td>10-4</td>
</tr>
<tr>
<td>10-3</td>
<td>Temp-table defined in af\app\logintt.i</td>
<td>10-5</td>
</tr>
<tr>
<td>10-4</td>
<td>Temp-table defined in af\app\afgtglobalctrl.i</td>
<td>10-5</td>
</tr>
<tr>
<td>10-5</td>
<td>Temp-tables defined in af\app\afgtpersist.i</td>
<td>10-5</td>
</tr>
<tr>
<td>10-6</td>
<td>Other temp-tables used by the Session Manager</td>
<td>10-6</td>
</tr>
<tr>
<td>11-1</td>
<td>User Interface Manager files</td>
<td>11-2</td>
</tr>
<tr>
<td>12-1</td>
<td>Web Request Manager files</td>
<td>12-2</td>
</tr>
</tbody>
</table>
Preface

This Preface contains the following sections:

- Purpose
- Audience
- Organization
- Using this manual
- Public and private APIs
- Typographical conventions
- OpenEdge messages
- Third party acknowledgements
Purpose

This manual provides a basic reference to the APIs that a developer uses to make calls to the Progress Dynamics® environment managers. Progress Dynamics is the Progress® OpenEdge® commercialization of the ongoing Internet Component Framework (ICF) project of the Progress Open Source Software Exchange (Progress POSSE, www.possenet.org).

Audience

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

Organization

This 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.

Using 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 dynamics directory, most of the paths are relative to DLC\src\dynamics. The paths for files that are not under the dynamics directory are given relative to DLC\src.

OpenEdge provides a special purpose programming language for building business applications. In the documentation, the formal name for this language is ABL (Advanced Business Language). With few exceptions, all keywords of the language appear in all UPPERCASE, using a font that is appropriate to the context. All other alphabetic language content appears in mixed case.

For the latest documentation updates see the OpenEdge Product Documentation category on PSDN http://www.psdn.com/library/kbcategory.jspa?categoryID=129.

References to ABL compiler and run-time features

ABL is both a compiled and interpreted language that executes in a run-time engine that the documentation refers to as the ABL Virtual Machine (AVM). When documentation refers to ABL source code compilation, it specifies ABL or the compiler as the actor that manages compile-time features of the language. When documentation refers to run-time behavior in an executing ABL program, it specifies the AVM as the actor that manages the specified run-time behavior in the program.

For example, these sentences refer to the ABL compiler’s allowance for parameter passing and the AVM’s possible response to that parameter passing at run time: “ABL allows you to pass a dynamic temp-table handle as a static temp-table parameter of a method. However, if at run time the passed dynamic temp-table schema does not match the schema of the static temp-table parameter, the AVM raises an error.” The following sentence refers to run-time actions that the AVM can perform using a particular ABL feature: “The ABL socket object handle allows the AVM to connect with other ABL and non-ABL sessions using TCP/IP sockets.”
References to ABL data types

ABL provides built-in data types, pre-defined class data types, and user-defined class data types. References to built-in data types follow these rules:

- Like most other keywords, references to specific built-in data types appear in all UPPERCASE, using a font that is appropriate to the context. No uppercase reference ever includes or implies any data type other than itself.
- Wherever integer appears, this is a reference to the INTEGER or INT64 data type.
- Wherever decimal appears, this is a reference to the DECIMAL data type.
- Wherever numeric appears, this is a reference to the INTEGER, INT64, or DECIMAL data type.

References to pre-defined class data types appear in mixed case with initial caps, for example, Progress.Lang.Object. References to user-defined class data types appear in mixed case, as specified for a given application example.

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 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:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Italic typeface indicates the title of a document, or signifies new terms.</td>
</tr>
<tr>
<td><strong>SMALL, BOLD CAPITAL LETTERS</strong></td>
<td>Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, <em>GET</em> and <em>CTRL</em>.</td>
</tr>
<tr>
<td><strong>KEY1+KEY2</strong></td>
<td>A plus sign between key names indicates a simultaneous key sequence: you press and hold down the first key while pressing the second key. For example, <em>CTRL+X</em>.</td>
</tr>
<tr>
<td><strong>KEY1 KEY2</strong></td>
<td>A space between key names indicates a sequential key sequence: you press and release the first key, then press another key. For example, <em>ESCAPE H</em>.</td>
</tr>
</tbody>
</table>

Syntax:

| Fixed width                       | A fixed-width font is used in syntax statements, code examples, system output, and filenames.                                           |
| **Fixed-width italics**           | Fixed-width italics indicate variables in syntax statements.                                                                             |
| **Fixed-width bold**              | Fixed-width bold indicates variables with special emphasis.                                                                                |
| **UPPERCASE fixed width**         | Uppercase words are ABL keywords. Although these are always shown in uppercase, you can type them in either uppercase or lowercase in a procedure. |

OpenEdge messages

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

- **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.

- **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before 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 OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.
After displaying a message, OpenEdge 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 after execution messages.

- Returns to the Progress Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.

- Halts processing of a procedure and returns immediately to the Progress Procedure Editor. This does not happen often.

- Terminates the current session.

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

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

If you encounter an error that terminates OpenEdge, note the message number before restarting.

**Obtaining more information about OpenEdge messages**

In Windows platforms, use OpenEdge online help to obtain more information about OpenEdge messages. Many OpenEdge tools include the following Help menu options to provide information about messages:

- Choose Help → Recent Messages to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.

- Choose Help → Messages and then enter the message number to display a description of a specific OpenEdge message.

- In the Procedure Editor, press the HELP key or F1.

**Third party acknowledgements**


OpenEdge includes software developed by the Apache Software Foundation (http://www.apache.org/). Copyright © 1999 The Apache Software Foundation. All rights reserved (Xerces C++ Parser (XML) and Xerces2 Java Parser (XML)); Copyright © 1999-2002 The Apache Software Foundation. All rights reserved (Xerces Parser (XML); and Copyright © 2000-2003 The Apache Software Foundation. All rights reserved (Ant). The names “Apache,” “Xerces,” “ANT,” and “Apache Software Foundation” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “Apache”, nor may “Apache” appear in their name, without prior written permission of the Apache Software Foundation. For written permission, please contact apache@apache.org. Software distributed on an “AS IS” basis. WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific
language governing rights and limitations under the License agreement that accompanies the product.


OpenEdge includes software developed by Vermont Creative Software. Copyright © 1988-1991 by Vermont Creative Software.

OpenEdge includes code licensed from RSA Security, Inc. Some portions licensed from IBM are available at http://oss.software.ibm.com/icu4j/.

OpenEdge includes the UnixWare platform of Perl Runtime authored by Kiem-Phong Vo and David Korn. Copyright © 1991, 1996 by AT&T Labs. Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED “AS IS”, WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHORS NOR AT&T LABS MAKE ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

OpenEdge includes the RSA Data Security, Inc. MD5 Message-Digest Algorithm. Copyright ©1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

OpenEdge includes software developed by the World Wide Web Consortium. Copyright © 1994-2002 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All rights reserved. This work is distributed under the W3C® Software License [http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231] in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

OpenEdge includes Sonic software, which includes software developed by Apache Software Foundation (http://www.apache.org/). Copyright © 1999-2000 The Apache Software Foundation. All rights reserved. The names “Ant”, “Axis”, “Xalan,” “FOP,” “The Jakarta Project”, “Tomcat”, “Xerces” and/or “Apache Software Foundation” must not be used to endorse or promote products derived from the Product without prior written permission. Any product derived from the Product may not be called “Apache”, nor may “Apache” appear in their name, without prior written permission. For written permission, please contact apache@apache.org.

OpenEdge includes Sonic software, which includes the JMX Technology from Sun Microsystems, Inc. Use and Distribution is subject to the Sun Community Source License available at http://sun.com/software/communitysource.

OpenEdge includes Sonic software, which includes software developed by the ModelObjects Group (http://www.modelobjects.com). Copyright © 2000-2001 ModelObjects Group. All rights reserved. The name “ModelObjects” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “ModelObjects”, nor may “ModelObjects” appear in their name, without prior written permission. For written permission, please contact djacobs@modelobjects.com.
OpenEdge includes Sonic software, which includes files that are subject to the Netscape Public License Version 1.1 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/NPL/. Software distributed under the License is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is Mozilla Communicator client code, released March 31, 1998. The Initial Developer of the Original Code is Netscape Communications Corporation. Portions created by Netscape are Copyright © 1998-1999 Netscape Communications Corporation. All Rights Reserved.

OpenEdge includes software Copyright © 2003-2006, Terence Parr All rights reserved. Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes ICU software 1.8 and later - Copyright © 1995-2003 International Business Machines Corporation and others All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

OpenEdge includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). Copyright © 1998-2007 The OpenSSL Project. All rights reserved. This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com). Copyright © 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project. Software distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes Sonic software which includes a version of the Saxon XSLT and XQuery Processor from Saxonica Limited that has been modified by Progress Software Corporation. The contents of the Saxon source code and the modified source code file (Configuration.java) are subject to the Mozilla Public License Version 1.0 (the “License”); you may not use these files except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/MPL/ and a copy of the license (MPL-1.0.html) can also be found in the installation directory, in the Docs7.5/third_party_licenses folder, along with a copy of the modified code (Configuration.java); and a description of the modifications can be found in the Progress SonicMQ and Progress Sonic ESB v7.5 README files. Software distributed under the License is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is The SAXON XSLT and XQuery Processor from Saxonica Limited. The Initial Developer of the Original Code is Michael Kay
OpenEdge includes software developed by IBM. Copyright © 1995-2003 International Business Machines Corporation and others. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. Software distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

OpenEdge includes Sonic software, which includes software developed by ExoLab Project (http://www.exolab.org/). Copyright © 2000 Intalio Inc. All rights reserved. The names “Castor” and/or “ExoLab” must not be used to endorse or promote products derived from the Products without prior written permission. For written permission, please contact info@exolab.org. Exolab, Castor and Intalio are trademarks of Intalio Inc.

OpenEdge includes Sonic software, which includes software Copyright © 1999 CERN - European Organization for Nuclear Research. Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. CERN makes no representations about the suitability of this software for any purpose. It is provided "as is" without expressed or implied warranty.

OpenEdge includes Sonic software, which includes software developed by the University Corporation for Advanced Internet Development http://www.ucaid.edu Internet2 Project. Copyright © 2002 University Corporation for Advanced Internet Development, Inc. All rights reserved. Neither the name of OpenSAML nor the names of its contributors, nor Internet2, nor the University Corporation for Advanced Internet Development, Inc., nor UCAID may be used to endorse or promote products derived from this software and products derived from this software may not be called OpenSAML, Internet2, UCAID, or the University Corporation for Advanced Internet Development, nor may OpenSAML appear in their name without prior written permission of the University Corporation for Advanced Internet Development. For written permission, please contact opensaml@opensaml.org.

OpenEdge includes DataDirect products for the Microsoft SQL Server database which contains a licensed implementation of the Microsoft TDS Protocol.

OpenEdge includes Sonic software, which includes code licensed from Mort Bay Consulting Pty. Ltd. The Jetty Package is Copyright © 1998 Mort Bay Consulting Pty. Ltd. (Australia) and others.
Configuration File Manager

The Configuration File Manager handles the initial startup of the Progress Dynamics® environment, as described in the following sections:

- Overview
- Temp-tables used by Configuration File Manager
- Configuration File Manager APIs
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>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Main code</td>
<td><code>af\app\afxmlcfgp.p</code></td>
</tr>
<tr>
<td>Included files</td>
<td><code>adecomm\_adetool.i</code></td>
</tr>
<tr>
<td></td>
<td><code>adm2\globals.i</code></td>
</tr>
<tr>
<td></td>
<td><code>af\sup2\aficfcheck.i</code></td>
</tr>
<tr>
<td></td>
<td><code>af\sup2\aftnode.i</code></td>
</tr>
<tr>
<td></td>
<td><code>af\sup2\afxmlcfgtt.i</code></td>
</tr>
<tr>
<td></td>
<td><code>af\sup2\afxmlreplctrl.i</code></td>
</tr>
<tr>
<td></td>
<td><code>ry\app\ryplipmain.i</code></td>
</tr>
<tr>
<td></td>
<td><code>ry\app\ryplipkill.i</code></td>
</tr>
<tr>
<td></td>
<td><code>ry\app\ryplipsetu.i</code></td>
</tr>
<tr>
<td></td>
<td><code>ry\app\ryplipshut.i</code></td>
</tr>
</tbody>
</table>

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in *OpenEdge Development: Progress Dynamics Advanced Development*. 
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 `afxmlcfgtt.i`. The preprocessors included in the names of certain tables allow you to easily incorporate temp-tables for custom managers.

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
</table>
| ttManager{&ttManagerExt} | 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) |
| ttService{&ttServiceExt} | cSessionType (Character)  
iOrder (Integer)  
cServiceType (Character)  
cServiceName (Character)  
cPhysicalService (Character)  
cConnectParams (Character)  
lDefaultService (Logical)  
lConnectAtStartup (Logical)  
lCanRunLocal (Logical)  
iStartOrder (Integer)  
iDBOrder (Integer)  
iCFGOrder (Integer)  
lUpdated (Logical)  
lDelete (Logical) |
| ttSession{&ttSessionExt} | cSessionType (Character)  
{&session-table-fields} |
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\afttnode.i

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttNode</td>
<td>cNode (Character)</td>
</tr>
<tr>
<td></td>
<td>cValue (Character)</td>
</tr>
<tr>
<td></td>
<td>iNodeLevel (Integer)</td>
</tr>
<tr>
<td></td>
<td>lDelete (Logical)</td>
</tr>
</tbody>
</table>

Configuration File Manager APIs

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

**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\afxmlcfgp.p

**Parameters:**

INPUT pcFileName AS CHARACTER

**Returns:** CHARACTER

**Notes:** None

**Examples:** See the validateConfigFile procedure in af\app\afxmlcfgp.p.
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\afxmlcfgp.p
Parameters:

INPUT pcString AS CHARACTER

Returns: CHARACTER
Notes: None
Examples: See the propertyExpander procedure in af\app\afxmlcfgp.p.

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.

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.
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.

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\afxmlcfgp.p

**Parameters:**

INPUT pcManagerName AS CHARACTER

**Returns:** HANDLE

**Notes:** None

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

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\afxmlcfgp.p

**Parameters:** None

**Returns:** CHARACTER

**Notes:** None

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

**getSessionParam**

This function returns the value associated with an option.

**Location:** af\app\afxmlcfgp.p

**Parameters:**

INPUT pcOption AS CHARACTER

**Returns:** CHARACTER

**Notes:** None

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

**initializeSession**

This procedure controls setting all the global environment settings.

**Location:** af\app\afxmlcfgp.p

**Parameters:**

INPUT pcICFParam AS CHARACTER

**Notes:** None

**Examples:** See the main block in dynamics\icfstart.p.
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\afxmlcfgp.p
Parameters: None
Returns: LOGICAL
Notes: None
Examples: See the start-super-proc procedure in af\sup2\afsrvtype.i.

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\afxmlcfgp.p
Parameters: None
Returns: LOGICAL
Notes: This function assumes that opaque handles are switched on.
Examples: See the main block in dynamics\icfstart.p.

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\afxmlcfgp.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.
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\afxmlcfgp.p
Parameters: None
Notes: None
Examples: See the initializeSession procedure in af\app\afxmlcfgp.p.

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\afxmlcfgp.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\afxmlcfgp.p.

propertyExpander

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

Location: af\app\afxmlcfgp.p
Parameters: None
Notes: None
Examples: See the initializeSession procedure in af\app\afxmlcfgp.p.
**sessionShutdown**

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

**Location:** af\app\afxmlcfgp.p  
**Parameters:** None  
**Notes:** None  
**Examples:** See the main block in af\app\afxmlcfgp.p.

**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\afxmlcfgp.p  
**Parameters:**  
INPUT plRunning AS LOGICAL  
**Returns:** LOGICAL  
**Notes:** None

**setSessionParam**

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

**Location:** af\app\afxmlcfgp.p  
**Parameters:**  
INPUT pcOption AS CHARACTER  
INPUT pcValue AS CHARACTER  
**Returns:** LOGICAL  
**Notes:** None  
**Examples:** See the initializeSession procedure in af\app\afxmlcfgp.p.
subscribeAll

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

**Location:**  
af\app\afxmlcfgp.p

**Parameters:**

INPUT phSourceProc AS HANDLE  
INPUT phTargetProc AS HANDLE

**Notes:**  
None

**Examples:**  
See the main block in dynamics\icfstart.p.
Connection and Service Type Managers

The Connection and Service Type Managers handle the connections to databases, AppServers™ for OpenEdge applications, and other services for the Progress Dynamics® environment, as outlined in the following sections:

- Overview
- Temp-tables used by Connection and Service Type Managers
- Connection Manager APIs
- AppServer Connection Manager APIs
- Database Connection Manager APIs
- WebService Service Type Manager APIs
Connection and Service Type Managers

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:

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

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Main code</td>
<td>af\app\afconmgrp.p</td>
</tr>
<tr>
<td>Included files</td>
<td>af\sup2\afglobs.i</td>
</tr>
<tr>
<td></td>
<td>af\app\afconttdef.i</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Main code</td>
<td>\af\app\afasconmgrp.p</td>
</tr>
<tr>
<td>Included files</td>
<td>\adecomm_adetool.i</td>
</tr>
<tr>
<td></td>
<td>\adecomm\appsrvtt.i</td>
</tr>
<tr>
<td></td>
<td>\af\sup2\aficfcheck.i</td>
</tr>
<tr>
<td></td>
<td>\af\sup2\afsrvtype.i</td>
</tr>
<tr>
<td>Other important files</td>
<td>\af\app\afservicetype.p</td>
</tr>
</tbody>
</table>

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

Table 2–3: Database Connection Manager files

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Main code</td>
<td>\af\app\afdbconmgrp.p</td>
</tr>
<tr>
<td>Included files</td>
<td>\af\sup2\aficfcheck.i</td>
</tr>
<tr>
<td></td>
<td>\af\sup2\afsrvtype.i</td>
</tr>
<tr>
<td>Other important files</td>
<td>\af\app\afservicetype.p</td>
</tr>
</tbody>
</table>

Table 2–4 shows the files that contain the WebService Connection Manager’s code.

Table 2–4: WebService Service Type Manager files

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Main code</td>
<td>\af\app\afwebconmgrp.p</td>
</tr>
<tr>
<td>Included files</td>
<td>\af\sup2\afsrvtype.i</td>
</tr>
<tr>
<td>Other important files</td>
<td>\af\app\afservicetype.p</td>
</tr>
</tbody>
</table>

For more information on these Managers, see the chapter on using the Progress Dynamics Managers in *OpenEdge Development: Progress Dynamics Advanced Development*. 
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 2–5.

Table 2–5: Temp-tables defined in af\app\afconttdef.i

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tt{&amp;ttPrefix}Service</td>
<td>cServiceName (Character)</td>
</tr>
<tr>
<td></td>
<td>cPhysicalService (Character)</td>
</tr>
<tr>
<td></td>
<td>cServiceType (Character)</td>
</tr>
<tr>
<td></td>
<td>cConnectParams (Character)</td>
</tr>
<tr>
<td></td>
<td>lDefaultService (Logical)</td>
</tr>
<tr>
<td>tt{&amp;ttPrefix}ServiceType</td>
<td>cServiceType (Character)</td>
</tr>
<tr>
<td></td>
<td>cSTProcName (Character)</td>
</tr>
<tr>
<td></td>
<td>hSTManager (Handle)</td>
</tr>
<tr>
<td></td>
<td>lUseHandle (Logical)</td>
</tr>
<tr>
<td></td>
<td>lConnectAtStartup (Logical)</td>
</tr>
</tbody>
</table>

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 2–6.

Table 2–6: Temp-tables defined in af\sup2\afsrvtype.i

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttService</td>
<td>cServiceName (Character)</td>
</tr>
<tr>
<td></td>
<td>cPhysicalService (Character)</td>
</tr>
<tr>
<td></td>
<td>cConnectParams (Character)</td>
</tr>
<tr>
<td></td>
<td>cServiceHandle (Character)</td>
</tr>
<tr>
<td></td>
<td>lDefaultService (Logical)</td>
</tr>
<tr>
<td></td>
<td>lConnectAtStartup (Logical)</td>
</tr>
<tr>
<td></td>
<td>{&amp;ServiceTypeFields}</td>
</tr>
</tbody>
</table>
Connection Manager APIs

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

**connectService**

This procedure establishes the connection to the required physical service.

**Location:** af\app\afconmgrp.p

**Parameters:**

- INPUT pcServiceName AS CHARACTER
- OUTPUT pcHandle AS CHARACTER

**Notes:** This is a wrapper procedure that calls `connectServiceWithParams` supplying blank values for the pcParameterList and pcSubstitutionList parameters.

**Examples:** See the `initializeServices` procedure in af\app\afconmgrp.p.

**connectServiceWithParams**

This procedure establishes the connection to the required physical service. The procedure accepts the substitute parameter list.

**Location:** af\app\afconmgrp.p

**Parameters:**

- INPUT pcServiceName AS CHARACTER
- INPUT pcParameterList AS CHARACTER
- INPUT pcSubstitutionList AS CHARACTER
- OUTPUT pcHandle AS CHARACTER

**Notes:** None
disconnectService
This procedure disconnects a physical service.

Location: af\app\afconmgrp.p
Parameters:
INPUT pcServiceName AS CHARACTER

Notes: None
Examples: See the processDB procedure in install\prc\inuiutilp.p.

getConnectionParams
This function returns the connection parameters for a logical service.

Location: af\app\afconmgrp.p
Parameters:
INPUT pcServiceName AS CHARACTER

Returns: CHARACTER
Notes: None
Examples: See the getConnectionString function in af\app\afservicetype.p.

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

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
**getServiceImplHandle**

This function gets the handle to a specific service.

*Location:* af\app\afconmgrp.p  
*Parameters:*  
INPUT pcServiceName AS CHARACTER  
*Returns:* CHARACTER  
*Notes:* None

**getServiceList**

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

*Location:* af\app\afconmgrp.p  
*Parameters:*  
INPUT pcServiceType AS CHARACTER  
*Returns:* CHARACTER  
*Notes:* None  
*Examples:* See the `initializeObject` procedure in af\obj2\afgendatov.w.

**getServiceSTManager**

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

*Location:* af\app\afconmgrp.p  
*Parameters:*  
INPUT pcServiceName AS CHARACTER  
*Returns:* HANDLE  
*Notes:* None  
*Examples:* See the `isConnected` function in af\app\afconmgrp.p.

**getServiceType**

This function returns the service type of a known service.

*Location:* af\app\afconmgrp.p  
*Parameters:*  
INPUT pcServiceName AS CHARACTER  
*Returns:* CHARACTER  
*Notes:* None
Connection and Service Type Managers

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.

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 plConnect AS LOGICAL  

**Notes:** None  
**Examples:** See the initializeSession procedure in af\app\afxmlcfgp.p.

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.
**isServiceRegistered**

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

**Location:** af\app\afconmgrp.p  
**Parameters:**  
INPUT pcServiceName AS CHARACTER  
**Returns:** LOGICAL  
**Notes:** None  
**Examples:** See the registerService procedure in af\app\afconmgrp.p.

**obtainConnectionTables**

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

**Location:** af\app\afconmgrp.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\afapppingp.p.

**plipShutdown**

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

**Location:** af\app\afconmgrp.p  
**Parameters:** None  
**Notes:** None
**reconnectService**

This procedure attempts to reconnect a service after it fails.

**Location:** af\app\afconmgrp.p

**Parameters:**

INPUT pcServiceName AS CHARACTER
OUTPUT pcHandle AS CHARACTER

**Notes:** None

**registerService**

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

**Location:** af\app\afconmgrp.p

**Parameters:**

INPUT phService AS HANDLE

**Notes:** None

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

**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\afconmgrp.p

**Parameters:**

INPUT phServiceProc AS HANDLE

**Returns:** LOGICAL

**Notes:** None

**Examples:** See the initializeSession procedure in af\app\afxmlcfgp.p.
AppServer Connection Manager APIs

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

appServerConnect

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

Location: af\app\afasconmgrp.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

appServerDisconnect

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

Location: af\app\afasconmgrp.p
Parameters:
  INPUT partition_name AS CHARACTER
Notes: The AppServer should only be disconnected with disconnectService.

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\afasconmgrp.p
Parameters:
  INPUT pcServiceName AS CHARACTER
  OUTPUT pcHandle AS CHARACTER
Notes: This is a wrapper procedure that calls connectServiceWithParams supplying blank values for the pcParameterList and pcSubstitutionList parameters.
Examples: See the appServerConnect procedure in af\app\afasconmgrp.p.
connectServiceWithParams

This procedure establishes the connection to the required physical service. The procedure accepts the substitute parameter list.

**Location:** af\app\afconmgrp.p

**Parameters:**

- INPUT pcServiceName AS CHARACTER
- INPUT pcParameterList AS CHARACTER
- INPUT pcSubstituteList AS CHARACTER
- OUTPUT pcHandle AS CHARACTER

**Notes:** None

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

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.
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

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

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
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

getJMSPartitions

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

Location: af\app\afascommgrp.p

Parameters: None

Returns: CHARACTER

Notes: None

getJMSPtnInfo

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

Location: af\app\afascommgrp.p

Parameters:

INPUT cPartition AS CHARACTER

Returns: CHARACTER

Notes: None

getPartitionsByType

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

Location: af\app\afascommgrp.p

Parameters:

INPUT cType AS CHARACTER

Returns: CHARACTER

Notes: None

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

**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

**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
**getServiceImpl**

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

**initializeSelf**

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

**Location:** `af\app\afasconmgrp.p`

**Parameters:** None

**Notes:** None

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

**isConnected**

This function determines whether the requested service type is connected.

**Location:** `af\app\afasconmgrp.p`

**Parameters:**

INPUT `pcServiceName` AS CHARACTER

**Returns:** LOGICAL

**Notes:** None

**Examples:** See the `connectService` procedure in `af\app\afasconmgrp.p`.

**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
loadPartitionInfo

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

parseConnectionParams

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

Location: af\app\afasconmgrp.p
Parameters:
INPUT pcParams AS CHARACTER
Returns: CHARACTER
Notes: None
Examples: See the registerService procedure in af\app\afasconmgrp.p.

plipShutdown

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

Location: af\app\afasconmgrp.p
Parameters: None
Notes: None

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
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 contains the service.

Notes: None

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

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 lCancel AS LOGICAL

Notes: None

setServiceHandle

This function sets the value of a 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
Database Connection Manager APIs

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

**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\afdbconmgrp.p

**Parameters:**

- INPUT pcServiceName AS CHARACTER
- OUTPUT pcHandle AS CHARACTER

**Notes:** This is a wrapper procedure that calls connectServiceWithParams supplying blank values for the pcParameterList and pcSubstitutionList parameters.

**connectServiceWithParams**

This procedure establishes the connection to the required physical service. The procedure accepts the substitute parameter list.

**Location:** af\app\afdbconmgrp.p

**Parameters:**

- INPUT pcServiceName AS CHARACTER
- INPUT pcParameterList AS CHARACTER
- INPUT pcSubstitutionList AS CHARACTER
- OUTPUT pcHandle AS CHARACTER

**Notes:** None

**disconnectService**

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

**Location:** af\app\afdbconmgrp.p

**Parameters:**

- INPUT pcServiceName AS CHARACTER

**Notes:** None
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

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

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\afdbconmgrp.p

**Parameters:**

INPUT pcServiceName AS CHARACTER

The logical service name.

**Returns:** CHARACTER

**Notes:** None
**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

**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

**getServiceHandle**

This function returns the handle to a specific service name.

**Location:** `af\app\afservicetype.p`

**Parameters:**

- **INPUT pcServiceName AS CHARACTER**
  - The logical service name.

**Returns:** CHARACTER

**Notes:** None
**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

**isConnected**

This function determines whether the requested service type is connected.

**Location:** af\app\afdbconmgrp.p  
**Parameters:**  
  - INPUT pcServiceName AS CHARACTER  
**Returns:** LOGICAL  
**Notes:** None  
**Examples:** See the connectService procedure in af\app\afdbconmgrp.p.

**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
parseConnectionParams

This function parses pcParams and returns the connection string.

Location: af\app\afdbconmgrp.p
Parameters:
   INPUT pcParams AS CHARACTER
Returns: CHARACTER
Notes: None
Examples: See the getConnectionString procedure in af\app\afservicetype.p.

plipShutdown

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

Location: af\app\afdbconmgrp.p
Parameters: None
Notes: None

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 contains the service.
Notes: None
**setServiceHandle**

This function sets the value of a 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

---

**WebService Service Type Manager APIs**

This section lists the APIs that you can use with the WebService Service Type Manager.

**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\afwebcommgr.p

**Parameters:**

- INPUT pcServiceName AS CHARACTER

- OUTPUT pcHandle AS CHARACTER

**Notes:** This is a wrapper procedure that calls connectServiceWithParams supplying blank values for the pcParameterList and pcSubstitutionList parameters.
connectServiceWithParams

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

The procedure retrieves the connection string and appends anything in the parameter list to the end of the connection string. It then invokes substituteParam in the target procedure to substitute arguments in the connection string with values from the substitution list. Finally it binds to the Web service using the updated connection string and returns the server object handle.

**Location:** af\app\afwebconmgrp.p

**Parameters:**

- INPUT pcServiceName AS CHARACTER
- INPUT pcParameterList AS CHARACTER
- INPUT pcSubstitutionList AS CHARACTER
- OUTPUT pcHandle AS CHARACTER

**Notes:** None

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

disconnectService

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

**Location:** af\app\afwebconmgrp.p

**Parameters:**

- INPUT pcServiceName AS CHARACTER

**Notes:** None

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

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

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
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

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

getServiceHandle

This function returns the handle to a specific service name.

**Location:**  af\app\afservicetype.p

**Parameters:**

INPUT pcServiceName AS CHARACTER

The logical service name.

**Returns:**  CHARACTER

**Notes:**  None
**getServiceImpl**

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

**isConnected**

This function determines whether the requested Web service is bound.

- **Location:** af\app\afwebconmgrp.p
- **Parameters:**
  - INPUT pcServiceName AS CHARACTER
- **Returns:** LOGICAL
- **Notes:** None
- **Examples:** See the connectServiceWithParams procedure in af\app\afwebconmgrp.p.

**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

**parseConnectionParams**

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

- **Location:** af\app\afwebconmgrp.p
- **Parameters:**
  - INPUT pcParams AS CHARACTER
- **Returns:** CHARACTER
- **Notes:** None
plipShutdown

This procedure cleans up the Web service connections when the WebService Service Type Manager shuts down.

Location: \af\app\afwebconmgr.p
Parameters: None
Notes: None

reconnectService

This procedure attempts to reconnect a Web service after the connection fails.

Location: \af\app\afwebconmgr.p
Parameters:

INPUT pcServiceName AS CHARACTER
OUTPUT pcHandle AS CHARACTER

Notes: This is a wrapper procedure that calls reconnectServiceWithParams supplying blank values for the pcParameterList and pcSubstitutionList parameters.

reconnectServiceWithParams

This procedure attempts to reconnect a Web service after the connection fails. It first calls disconnectService to ensure that the service is fully disconnected. It then calls reconnectServiceWithParams to reconnect to the service.

Location: \af\app\afwebconmgr.p
Parameters:

INPUT pcServiceName AS CHARACTER
INPUT pcParameterList AS CHARACTER
INPUT pcSubstitutionList AS CHARACTER
OUTPUT pcHandle AS CHARACTER

Notes: None
**setServiceHandle**

This function sets the value of a 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, as outlined in the following sections:

- Overview
- Customization Manager APIs
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 sets the global variable, server-side, to either YES for the server-side version or NO for the client-side version. Both programs then include the main code of the manager from the ry\app\rcusmngrp.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>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>ry\app\rcusclntp.p</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>ry\app\rcussrvrp.p</td>
</tr>
<tr>
<td>Main code</td>
<td>ry\app\rcusmngrp.i</td>
</tr>
<tr>
<td>Included files</td>
<td>adm2\globals.i</td>
</tr>
<tr>
<td></td>
<td>ry\app\rydefrescd.i</td>
</tr>
</tbody>
</table>

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in *OpenEdge Development: Progress Dynamics Advanced Development*. 
Customization Manager APIs

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

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\rycusmngrp.i
Parameters: None
Returns: CHARACTER
Notes: None
Examples: See the main block in ry\app\rygetmensp.p.

getCustomisationTypesPrioritised

This function returns the prioritized customization types for a session.

Location: ry\app\rycusmngrp.i
Parameters: None
Returns: CHARACTER
Notes: None
Examples: See the main block in af\app\apppingp.p.

getReferenceLanguage

This function returns the current reference for the language.

Location: ry\app\rycusmngrp.i
Parameters: None
Returns: CHARACTER
Notes: None
**getReferenceLoginCompany**

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

- **Location:** `ry\app\rycusmngp.i`
- **Parameters:** None
- **Returns:** CHARACTER
- **Notes:** None

**getReferenceSystem**

This function returns the current reference for the system.

- **Location:** `ry\app\rycusmngp.i`
- **Parameters:** None
- **Returns:** CHARACTER
- **Notes:** None

**getReferenceUIType**

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

- **Location:** `ry\app\rycusmngp.i`
- **Parameters:** None
- **Returns:** CHARACTER
- **Notes:** None

**getReferenceUser**

This function returns the current reference for the user.

- **Location:** `ry\app\rycusmngp.i`
- **Parameters:** None
- **Returns:** CHARACTER
- **Notes:** None
getReferenceUserCategory

This function returns the current reference for the user category.

**Location:** ry\app\rycusmngrp.i  
**Parameters:** None  
**Returns:** CHARACTER  
**Notes:** Since the information is only available on the server, this call cannot run on the client.

getSessionCustomisationReferences

This function returns the session customization references.

**Location:** ry\app\rycusmngrp.i  
**Parameters:** None  
**Returns:** CHARACTER  
**Notes:** None  
**Examples:** See the main block in af\app\appnngp.p.

getSessionResultCodes

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

**Location:** ry\app\rycusmngrp.i  
**Parameters:** None  
**Returns:** CHARACTER  
**Notes:** None  
**Examples:** See the main block in af\app\appnngp.p.

InitializeObject

This procedure subscribes the Customization Manager to the ICFCFM_LoginComplete event.

**Location:** ry\app\rycusmngrp.i  
**Parameters:** None  
**Notes:** None
**PlipShutdown**

This procedure closes any super procedures of the Customization Manager.

**Location:** \ry\app\rycusmngp.i

**Parameters:** None

**Notes:** None

**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\rycusmngp.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\rycusmngp.i.

**setSessionResultCodes**

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

**Location:** \ry\app\rycusmngp.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\rycusmngp.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, as outlined in the following sections:

- Overview
- Temp-tables used by General Manager
- General Manager APIs
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 sets the global variable, server-side, to either YES for the server-side version or NO for the client-side version. Both programs then include the main code of the manager from the af\app\afgenmngrp.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

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>af\app\afgenclntrp.p</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>af\app\afgensrvrp.p</td>
</tr>
<tr>
<td>Main code</td>
<td>af\app\afgenmngrp.i</td>
</tr>
<tr>
<td>Included files</td>
<td>af\app\afgencchentmapp.i</td>
</tr>
<tr>
<td></td>
<td>af\app\afgensgetemnnpp.i</td>
</tr>
<tr>
<td></td>
<td>af\app\afengsgetgcdp.i</td>
</tr>
<tr>
<td></td>
<td>af\app\gsmstttcch.i</td>
</tr>
<tr>
<td></td>
<td>af\app\gstenmn.i</td>
</tr>
<tr>
<td></td>
<td>af\sup2\afcheckerr.i</td>
</tr>
<tr>
<td></td>
<td>af\sup2\aferrortxt.i</td>
</tr>
<tr>
<td></td>
<td>af\sup2\afrun2.i</td>
</tr>
</tbody>
</table>

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in *OpenEdge Development: Progress Dynamics Advanced Development*. 
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 afapp\afgenmngrp.i

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttUser</td>
<td>user_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>user_category_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>user_full_name (Character)</td>
</tr>
<tr>
<td></td>
<td>user_login_name (Character)</td>
</tr>
<tr>
<td></td>
<td>user_creation_date (Date)</td>
</tr>
<tr>
<td></td>
<td>user_creation_time (Integer)</td>
</tr>
<tr>
<td></td>
<td>profile_user (Logical)</td>
</tr>
<tr>
<td></td>
<td>created_from_profile_user_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>external_userid (Integer)</td>
</tr>
<tr>
<td></td>
<td>user_password (Character)</td>
</tr>
<tr>
<td></td>
<td>password_minimum_length (Integer)</td>
</tr>
<tr>
<td></td>
<td>password_preexpired (Logical)</td>
</tr>
<tr>
<td></td>
<td>password_fail_count (Integer)</td>
</tr>
<tr>
<td></td>
<td>password_fail_date (Date)</td>
</tr>
<tr>
<td></td>
<td>password_fail_time (Integer)</td>
</tr>
<tr>
<td></td>
<td>password_creation_date (Date)</td>
</tr>
<tr>
<td></td>
<td>password_creation_time (Integer)</td>
</tr>
<tr>
<td></td>
<td>password_expiry_date (Date)</td>
</tr>
<tr>
<td></td>
<td>password_expiry_time (Integer)</td>
</tr>
<tr>
<td></td>
<td>update_password_history (Logical)</td>
</tr>
<tr>
<td></td>
<td>check_password_history (Logical)</td>
</tr>
<tr>
<td></td>
<td>last_login_date (Date)</td>
</tr>
<tr>
<td></td>
<td>last_login_time (Integer)</td>
</tr>
<tr>
<td></td>
<td>disabled (Logical)</td>
</tr>
<tr>
<td></td>
<td>language_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>password_expiry_days (Integer)</td>
</tr>
<tr>
<td></td>
<td>maintain_system_data (Logical)</td>
</tr>
<tr>
<td></td>
<td>default_login_company_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>user_email_address (Character)</td>
</tr>
<tr>
<td></td>
<td>development_user (Logical)</td>
</tr>
<tr>
<td></td>
<td>security_group (Logical)</td>
</tr>
<tr>
<td></td>
<td>default_security_group (Logical)</td>
</tr>
<tr>
<td>ttFilterSetClause</td>
<td>filter_set_code (Character)</td>
</tr>
<tr>
<td></td>
<td>entity_list (Character)</td>
</tr>
<tr>
<td></td>
<td>buffer_list (Character)</td>
</tr>
<tr>
<td></td>
<td>additional_arguments (Character)</td>
</tr>
<tr>
<td></td>
<td>filter_set_clause (Character)</td>
</tr>
</tbody>
</table>
The General Manager also uses the temp-tables described in Table 4–3 to store information about entity display fields and entity mnemonics. These 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 a copy of the `ttEntityMnemonic` temp-table.

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

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttEntityDisplayField</td>
<td>entity_display_field_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>entity_mnemonic (Character)</td>
</tr>
<tr>
<td></td>
<td>display_field_name (Character)</td>
</tr>
<tr>
<td></td>
<td>display_field_order (Integer)</td>
</tr>
<tr>
<td></td>
<td>display_field_label (Character)</td>
</tr>
<tr>
<td></td>
<td>display_field_column_label (Character)</td>
</tr>
<tr>
<td></td>
<td>display_field_format (Character)</td>
</tr>
<tr>
<td></td>
<td>display_field_datatype (Character)</td>
</tr>
<tr>
<td>ttEntityMnemonic</td>
<td>entity_mnemonic (Character)</td>
</tr>
<tr>
<td>&amp; ttEntityMap</td>
<td>entity_mnemonic_short_desc (Character)</td>
</tr>
<tr>
<td></td>
<td>entity_mnemonic_description (Character)</td>
</tr>
<tr>
<td></td>
<td>auto_properform_strings (Logical)</td>
</tr>
<tr>
<td></td>
<td>entity_mnemonic_label_prefix (Character)</td>
</tr>
<tr>
<td></td>
<td>entity_mnemonic_obj (Character)</td>
</tr>
<tr>
<td></td>
<td>entity_description_field (Character)</td>
</tr>
<tr>
<td></td>
<td>entity_description_procedure (Character)</td>
</tr>
<tr>
<td></td>
<td>entity_narration (Character)</td>
</tr>
<tr>
<td></td>
<td>entity_object_field (Character)</td>
</tr>
<tr>
<td></td>
<td>table_has_object_field (Logical)</td>
</tr>
<tr>
<td></td>
<td>entity_key_field (Character)</td>
</tr>
<tr>
<td></td>
<td>table_prefix_length (Integer)</td>
</tr>
<tr>
<td></td>
<td>field_name_separator (Character)</td>
</tr>
<tr>
<td></td>
<td>auditing_enabled (Logical)</td>
</tr>
<tr>
<td></td>
<td>version_data (Logical)</td>
</tr>
<tr>
<td></td>
<td>deploy_data (Logical)</td>
</tr>
<tr>
<td></td>
<td>entity_dbname (Character)</td>
</tr>
<tr>
<td></td>
<td>replicate_entity_mnemonic (Character)</td>
</tr>
<tr>
<td></td>
<td>replicate_key (Character)</td>
</tr>
<tr>
<td></td>
<td>scm_field_name (Character)</td>
</tr>
<tr>
<td></td>
<td>reuse_deleted_keys (Logical)</td>
</tr>
<tr>
<td></td>
<td>hasAudit (Logical)</td>
</tr>
<tr>
<td></td>
<td>hasComment (Logical)</td>
</tr>
<tr>
<td></td>
<td>hasAutoComment (Logical)</td>
</tr>
</tbody>
</table>
Table 4–4 describes the temp-table defined in `af\app\gsmstttcch.i`, which is used to cache status information.

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttStatusCache</td>
<td>tStatusObj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>tCategoryType (Character)</td>
</tr>
<tr>
<td></td>
<td>tCategoryGroup (Character)</td>
</tr>
<tr>
<td></td>
<td>tCategorySubGroup (Character)</td>
</tr>
<tr>
<td></td>
<td>tStatusShortDesc (Character)</td>
</tr>
<tr>
<td></td>
<td>tStatusDescription (Character)</td>
</tr>
<tr>
<td></td>
<td>tStatusTLA (Character)</td>
</tr>
</tbody>
</table>

**General Manager APIs**

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

**checkIfOverlaps**

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

**Location:** `af\app\afgenmngrp.i`

**Parameters:**

```plaintext
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.

Note: It creates a buffer for pcTable. Thus if you passed in gsm_person, the created buffer will be bgsm_person. In pcAdditionalWhere remember to use the prefixed 'b' in your criteria specification.

OUTPUT plOverlap 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: None
Examples: See the main block in af\app\afgenchkifovrlpp.p.

**convertTimeToInteger**

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

Location: af\app\afgenmngrp.i

Parameters:

INPUT pcTime AS CHARACTER
  Can be specified as HH:MM, HH:MM:SS, or HHMMSS.

Returns: INTEGER

Notes: None
Examples: See the getDataValue function in ry\tem\rytemtimesdf.w.
createFolder

This function creates a folder if it does not exist.

Location: \af\app\afgenmngrp.i
Parameters:

INPUT pcFolderName AS CHARACTER

Returns: LOGICAL
Notes: None
Examples: See the main block in af\app\afgendlogp.p.

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\afgenmngrp.i
Parameters:

INPUT pcLastName AS CHARACTER
INPUT pcFirstName AS CHARACTER
INPUT pcInitials AS CHARACTER

Returns: CHARACTER
Notes: None

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\afgenmngrp.i
Parameters:

INPUT plOnlyUseCache 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\afgentdbfrimpentp.p.

**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\afgenmngrp.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\afapppingp.p.

**getDumpName**

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

Location: af\app\afgenmngrp.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\afgenmngrp.i.
**getEntityCacheBuffer**

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

**Location:**   
`af\app\afgenmngrp.i`

**Parameters:**

INPUT `pcEntity` AS CHARACTER  
INPUT `pcTableName` AS CHARACTER

**Returns:**   
HANDLE

**Notes:**   
None

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

**getEntityDescription**

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

**Location:**   
`af\app\afgenmngrp.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\afgenmngrp.i`. 
**getEntityDetail**

This procedure returns information about an entity.

**Location:** af\app\afgenmngrp.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\gsmcmfullo.w.

**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\afgenmngrp.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\gsmcmviewv.w.
**getEntityExists**

This procedure checks that the specified record exists.

**Location:** `af\app\afgenmnggrp.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\afgententestp.p`.

**getEntityFieldCacheBuffer**

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

**Location:** `af\app\afgenmnggrp.i`

**Parameters:**

- INPUT `pcEntity` AS CHARACTER
- INPUT `pcTableName` AS CHARACTER

**Returns:** HANDLE

**Notes:** None

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

**getEntityTableName**

This procedure returns the table name based on an entity.

**Location:** `af\app\afgenmnggrp.i`

**Parameters:**

- INPUT `pcEntityMnemonic` AS CHARACTER
- INPUT `pcLogicalDBName` AS CHARACTER
- OUTPUT `pcEntityTablename` AS CHARACTER

**Notes:** None

**Examples:** See the `buildQuery` procedure in `af\app\secgetdata.p`. 
**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\afgenmngrp.i`

**Parameters:**

- `pcCollationTableName` AS CHARACTER

**Returns:** CHARACTER

**Notes:** None

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

**getInternalEntries**

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

**Location:** `af\app\afgenmngrp.i`

**Parameters:** None

**Returns:** CHARACTER

**Notes:** None

**Examples:** See the `getInternalEntryExists` in `af\app\afsesmngrp.i`.

**getKeyField**

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

**Location:** `af\app\afgenmngrp.i`

**Parameters:**

- `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\afgenmngrp.i`. 
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\afgenmngrp.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\afgengt1ngtxtp.p.
getNextSequenceValue

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

**Location:** af\app\afgenmngrp.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\gsmmfullo.w.

getObjField

This function retrieves the field that serves as the object (_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\afgenmngrp.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\afgenmngrp.i.
getOEMCode

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

Location: af\app\afgenmnggrp.i
Parameters:
INPUT pcEntityMnemonic AS CHARACTER
INPUT pdEntityObj AS DECIMAL
Returns: CHARACTER
Notes: Use the getOEMDescription function for a name or description field.

getOEMDescription

This function returns a name or similar description field.

Location: af\app\afgenmnggrp.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.

getPropertyFromList

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

Location: af\app\afgenmnggrp.i
Parameters:
INPUT pcPropertyList AS CHARACTER
INPUT pcPropertyName AS CHARACTER
Returns: CHARACTER
Notes: None
Examples: See the constructObject procedure in af\obj2\gsmcvieww.w.
**getRecordCheckAudit**

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

**Location:**  af\app\afgenmngrp.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

**getRecordCheckComment**

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

**Location:**  af\app\afgenmngrp.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 plRowCommentExist AS LOGICAL**

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

**OUTPUT pcRowCommentAuto AS CHARACTER**

**Notes:** None
**getRecordDetail**

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

**Location:** af\app\afgenmngrp.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\gsmcmviewv.w.

**getRecordUserProp**

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

**Location:** af\app\afgenmngrp.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.

**getSequenceConfirmation**

This function checks whether a specified sequence is valid.

**Location:** af\app\afgenmngrp.i

**Parameters:**

- INPUT pdCompanyObj AS DECIMAL
- INPUT pcEntityMnemonic AS CHARACTER
- INPUT pcSequenceTLA AS CHARACTER

**Returns:** LOGICAL

**Notes:** None
**getSequenceExist**

This procedure determines whether a sequence or reference exists.

*Location:* `af\app\afgenmngrp.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\afgenmngrp.i`.

**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\afgenmngrp.i`

*Parameters:*

- **INPUT** `pcSequenceFormat` AS CHARACTER
- **OUTPUT** `pcQuantityIndicator` AS CHARACTER
- **OUTPUT** `pcSequenceMask` AS CHARACTER

*Notes:* None

*Examples:* See the `getSequenceValue` procedure in `af\app\afgenmngrp.i`.

**getSiteNumber**

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

*Location:* `af\app\afgenmngrp.i`

*Parameters:*

- **OUTPUT** `piSite` AS INTEGER

  The current site number.

*Notes:* None

*Examples:* See the `getSequenceValue` procedure in `af\app\afgenmngrp.i`. 
**getStatusObj**

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

**Location:**  
`af\app\afgenmngrp.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.

**getStatusShortDesc**

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

**Location:**  
`af\app\afgenmngrp.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.

**getStatusTLA**

This function returns the status TLA to a calling procedure.

**Location:**  
`af\app\afgenmngrp.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.
getTableDumpName

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

**Location:** af\app\afgenmngrp.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\afgenmngrp.i.

ggetTableInfo

This procedure returns a CHR(4)-delimited list of the owning entity mnemonic, table name, and table key field for an updateable table in the supplied data source. The procedure returns the information in the following format:

```
<owning entity mnemonic> CHR(4) <table name> CHR(4) <table’s key field>
```

**Note:** A table’s key field is not necessarily the object (_obj) field.

**Location:** af\app\afgenmngrp.i

**Parameters:**

INPUT pcUpdatableTable AS CHARACTER

OUTPUT pcReturnValue AS CHARACTER

**Notes:** None

**Examples:** See the getUpdatableTableInfo function in af\app\afgenmngrp.i.
**getTableInfoObj**

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

**Location:** af\app\afgenmngrp.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\afgenmngrp.i.

**getUpdatableTableInfo**

This function returns a CHR(4)-delimited list of the owning entity mnemonic, table name, and table key field for an updateable table in the supplied data source. The procedure returns the information in the following format:

```
<owning entity mnemonic> CHR(4) <table name> CHR(4) <table's key field>
```

**Note:** A table’s key field is not necessarily the object (_obj) field.

**Location:** af\app\afgenmngrp.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.
**getUpdatableTableInfoObj**

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

**Location:**  af\app\afgenmngrp.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.

**getUserSourceLanguage**

This procedure returns the source language for a user.

**Location:**  af\app\afgenmngrp.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.

**haveOutstandingUpdates**

This function checks the ryt_dbupdate_status record for any outstanding updates.

**Location:**  af\app\afgenmngrp.i

**Parameters:**  None

**Returns:**  LOGICAL

**Notes:**  None

**Examples:**  See initializeSession in af\app\afxmlcfgp.p.
listLookup

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

Location: af\app\afgenmngrp.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

plipShutdown

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

Location: af\app\afgenmngrp.i

Parameters: None

Notes: This procedure currently contains no active code.

refreshMnemonicsCache

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

Location: af\app\afgenmngrp.i

Parameters: None

Notes: None

Examples: See the validateEntityMnemonic procedure in af\app\afgenmngrp.i.
**setPropertyValueInList**

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\afgenmmgrp.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.

**setWidgetAttribute**

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

**Location:**  
af\app\afgenmmgrp.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
updateTableViaSDO

This procedure updates the database through the relevant SDO.

**Location:** af\app\afgenmngp.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\rycoilog3p.p.
validateEntityMnemonic

This procedure validates a given entity mnemonic. It runs against the Repository's gsc_entity_mnemonic table first. If the entity mnemonic can be found in the gsc_entity_mnemonic table, then this procedure forces a refresh of the cache. If the entity mnemonic cannot be found in the database or the cache, then a blank value is returned. This procedure does not attempt to create an entity mnemonic record.

**Location:** af\app\afgenmngrp.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), as outlined in the following sections:

- Overview
- Temp-tables used by Localization Manager
- Localization Manager APIs
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 must 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 sets the global variable, server-side, to either YES for the server-side version or 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>af\app\aftrnclnntp.p</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>af\app\aftrnsrvrp.p</td>
</tr>
<tr>
<td>Main code</td>
<td>af\app\aftrnmngrp.i</td>
</tr>
<tr>
<td>Included files</td>
<td>adm2\tttranslate.i af\app\aftrnbldtrncp.i af\app\aftrngtmtrnp.i af\app\aftrnupdtrnp.i af\app\afttttranslation.i af\sup2\afcheckerr.i af\sup2\afglobals.i</td>
</tr>
</tbody>
</table>

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in *OpenEdge Development: Progress Dynamics Advanced Development*. 
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 af\app\aftttranslation.i

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttTranslation</td>
<td>translation_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>language_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>object_filename (Character)</td>
</tr>
<tr>
<td></td>
<td>widget_type (Character)</td>
</tr>
<tr>
<td></td>
<td>widget_name (Character)</td>
</tr>
<tr>
<td></td>
<td>widget_entry (Integer)</td>
</tr>
<tr>
<td></td>
<td>original_label (Character)</td>
</tr>
<tr>
<td></td>
<td>translation_label (Character)</td>
</tr>
<tr>
<td></td>
<td>original_tooltip (Character)</td>
</tr>
<tr>
<td></td>
<td>translation_tooltip (Character)</td>
</tr>
<tr>
<td></td>
<td>source_language_obj (Decimal)</td>
</tr>
</tbody>
</table>

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\tttranslate.i

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttTranslate</td>
<td>dLanguageObj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>dSourceLanguageObj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>cLanguageName (Character)</td>
</tr>
<tr>
<td></td>
<td>cObjectName (Character)</td>
</tr>
<tr>
<td></td>
<td>lGlobal (Logical)</td>
</tr>
<tr>
<td></td>
<td>lExtractedGlobal (Logical)</td>
</tr>
<tr>
<td></td>
<td>cWidgetType (Character)</td>
</tr>
<tr>
<td></td>
<td>cWidgetName (Character)</td>
</tr>
<tr>
<td></td>
<td>hWidgetHandle (Handle)</td>
</tr>
<tr>
<td></td>
<td>iWidgetEntry (Integer)</td>
</tr>
<tr>
<td></td>
<td>lDelete (Logical)</td>
</tr>
<tr>
<td></td>
<td>cTranslatedLabel (Character)</td>
</tr>
<tr>
<td></td>
<td>cOriginalLabel (Character)</td>
</tr>
<tr>
<td></td>
<td>cTranslatedTooltip (Character)</td>
</tr>
<tr>
<td></td>
<td>cOriginalTooltip (Character)</td>
</tr>
</tbody>
</table>
Localization Manager APIs

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

**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 `dynamics\icfstart.p`.

**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\aftemlognw.w.

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\aftrnmngrp.i
Parameters: None
Notes: None

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.
**Localization Manager**

```
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\aftrnmngrp.i`.

### multiTranslation

This procedure performs multiple translations in one pass. It then stores the results of the translations 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\aftrnmngrp.i`

**Parameters:**

- **INPUT plAllLanguages 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\afsesmngp.i.

plipShutdown

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

Location: af\app\aftrnmngp.i
Parameters: None
Notes: This procedure currently contains no active code.

receiveCacheClient

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

Location: af\app\aftrnmngp.i
Parameters:

INPUT TABLE FOR ttTranslation
INPUT pdLanguageObj AS DECIMAL

Notes: None
Examples: See the loginCacheAfter procedure in af\app\afsesmngp.i.
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\afmsgp.p.

translateSingleObject

This procedure returns the translated labels and ToolTips for a single object. This procedure is called from the Repository object retrieval process. It avoids the overhead of creating table entries, performing translations, reading through the translations, and then applying them to the object being retrieved.

Location: af\app\aftrnmngrp.i

Parameters:

INPUT pdLanguageObj AS DECIMAL

INPUT pcObjectName AS CHARACTER

INPUT pcWidgetName AS CHARACTER

INPUT pcWidgetType AS CHARACTER

INPUT piWidgetEntries AS INTEGER

OUTPUT pcLabels AS CHARACTER

OUTPUT pcTooltips AS CHARACTER

Notes: The passed in language_obj is assumed to be valid.

Examples: See the setFolderDetails procedure in ry\app\ryrepngrp.i.
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 plTranslated 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\aftrnmngrp.i
Parameters:

INPUT plTranslated 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.

updateTranslations

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

Location: af\app\aftrnmngrp.i
Parameters:

INPUT TABLE FOR ttTranslate
   The temp-table of translations.

Notes: None
Examples: See the updateRecord procedure in ry\obj\rydyncnv.w.
Profile Manager

The Profile Manager handles access to information on user preferences for the Progress Dynamics framework, as outlined in the following sections:

- Overview
- Temp-tables used by Profile Manager
- Profile Manager APIs
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 systemwide information, such as ToolTips enabled and similar UI options. The Profile Manager stores all of these settings by user. The user can decide 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 sets the global variable, server-side, to either YES for the server-side version or 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>af\sup2\afproclntp.p</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>af\app\afprosrvrp.p</td>
</tr>
<tr>
<td>Main code</td>
<td>af\app\afpromngrp.i</td>
</tr>
<tr>
<td>Included files</td>
<td>af\app\afbldclicp.p</td>
</tr>
<tr>
<td></td>
<td>af\app\afproupdcadbp.i</td>
</tr>
<tr>
<td></td>
<td>af\app\afttprofiledata.i</td>
</tr>
<tr>
<td></td>
<td>af\sup2\afcheckerr.i</td>
</tr>
<tr>
<td></td>
<td>af\sup2\aferrortxt.i</td>
</tr>
<tr>
<td></td>
<td>af\sup2\afglobals.i</td>
</tr>
</tbody>
</table>

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in *OpenEdge Development: Progress Dynamics Advanced Development*. 
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 af\app\afttprofiledata.i

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttProfileData</td>
<td>user_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>profile_type_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>profile_code_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>profile_data_key (Character)</td>
</tr>
<tr>
<td></td>
<td>context_id (Character)</td>
</tr>
<tr>
<td></td>
<td>profile_data_value (Character)</td>
</tr>
<tr>
<td></td>
<td>profile_data_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>cProfileTypeCode (Character)</td>
</tr>
<tr>
<td></td>
<td>cProfileCode (Character)</td>
</tr>
<tr>
<td></td>
<td>cAction (Character)</td>
</tr>
</tbody>
</table>

Profile Manager APIs

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

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 nonblank 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\afsesmngrp.i.
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 plCheckPermanentOnly AS LOGICAL**
  Check permanent data only flag, default = YES.

- **INPUT plCheckCacheOnly AS LOGICAL**
  Check client cache only, default = NO.

- **OUTPUT plExists AS LOGICAL**
  Does the profile data exist.

**Notes:**

- If the plCheckPermanentOnly flag is set to NO and there are database entries that exist for the session only, then the procedure returns TRUE.

- If the plCheckCacheOnly 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.

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 relogin procedure in af\app\afsesmngrp.i.
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 nonblank 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 dynamics/as_disconnect.p.

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 the 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 plNextRecordFlag 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\afgenmngp.i.

getProfileTTHandle

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

Location: af\app\afpromngp.i
Parameters: None
Returns: HANDLE
Notes: None
Examples: See the resetSesPerFlag procedure in af\sup2\afsdfiltw.w.

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\afpromngp.i
Parameters: None
Notes: None

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\afpromngp.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\afsesmngp.i.
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 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`.

**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 relogon procedure in `af\app\afsesmngrp.i`. 
The Referential Integrity Manager handles data versioning and supports the reuse of unique keys, as outlined in the following sections:

- **Overview**
- **Temp-tables used by Referential Integrity Manager**
- **Referential Integrity Manager API**
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.

To prevent clashes on unique key information, the Referential Integrity Manager checks the Repository `gst_record_version` table for information about old keys. If you attempt to re-create 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Main code</td>
<td><code>ry\app\ryrisrvrp.p</code></td>
</tr>
<tr>
<td>Included files</td>
<td><code>adm2\globals.i</code></td>
</tr>
<tr>
<td></td>
<td><code>af\sup2\afcheckerr.i</code></td>
</tr>
<tr>
<td></td>
<td><code>ry\app\ryplipkill.i</code></td>
</tr>
<tr>
<td></td>
<td><code>ry\app\ryplipmain.i</code></td>
</tr>
<tr>
<td></td>
<td><code>ry\app\ryplipsetu.i</code></td>
</tr>
<tr>
<td></td>
<td><code>ry\app\ryplipshut.i</code></td>
</tr>
</tbody>
</table>

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in *OpenEdge Development: Progress Dynamics Advanced Development*. 
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

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttEntity¹</td>
<td>entity_mnemonic (Character)</td>
</tr>
<tr>
<td></td>
<td>entity_mnemonic_short_desc (Character)</td>
</tr>
<tr>
<td></td>
<td>entity_mnemonic_description (Character)</td>
</tr>
<tr>
<td></td>
<td>auto_properform_strings (Logical)</td>
</tr>
<tr>
<td></td>
<td>entity_mnemonic_label_prefix (Character)</td>
</tr>
<tr>
<td></td>
<td>entity_mnemonic_obj (Character)</td>
</tr>
<tr>
<td></td>
<td>entity_description_field (Character)</td>
</tr>
<tr>
<td></td>
<td>entity_description_procedure (Character)</td>
</tr>
<tr>
<td></td>
<td>entity_narration (Character)</td>
</tr>
<tr>
<td></td>
<td>entity_object_field (Character)</td>
</tr>
<tr>
<td></td>
<td>table_has_object_field (Logical)</td>
</tr>
<tr>
<td></td>
<td>entity_key_field (Character)</td>
</tr>
<tr>
<td></td>
<td>table_prefix_length (Integer)</td>
</tr>
<tr>
<td></td>
<td>field_name_separator (Character)</td>
</tr>
<tr>
<td></td>
<td>auditing_enabled (Logical)</td>
</tr>
<tr>
<td></td>
<td>version_data (Logical)</td>
</tr>
<tr>
<td></td>
<td>deploy_data (Logical)</td>
</tr>
<tr>
<td></td>
<td>entity_dbname (Character)</td>
</tr>
<tr>
<td></td>
<td>replicate_entity_mnemonic (Character)</td>
</tr>
<tr>
<td></td>
<td>replicate_key (Character)</td>
</tr>
<tr>
<td></td>
<td>scm_field_name (Character)</td>
</tr>
<tr>
<td></td>
<td>reuse_deleted_keys (Logical)</td>
</tr>
<tr>
<td>tDSEntity²</td>
<td>dataset_entity_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>deploy_dataset_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>entity_sequence (Integer)</td>
</tr>
<tr>
<td></td>
<td>entity_mnemonic (Character)</td>
</tr>
<tr>
<td></td>
<td>primary_entity (Logical)</td>
</tr>
<tr>
<td></td>
<td>join_entity_mnemonic (Character)</td>
</tr>
<tr>
<td></td>
<td>join_field_list (Character)</td>
</tr>
<tr>
<td></td>
<td>filter_where_clause (Character)</td>
</tr>
<tr>
<td></td>
<td>delete_related_records (Logical)</td>
</tr>
<tr>
<td></td>
<td>overwrite_records (Logical)</td>
</tr>
<tr>
<td></td>
<td>keep_own_site_data (Logical)</td>
</tr>
<tr>
<td></td>
<td>use_relationship (Logical)</td>
</tr>
<tr>
<td></td>
<td>relationship_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>deletion_action (Character)</td>
</tr>
<tr>
<td></td>
<td>exclude_field_list (Character)</td>
</tr>
<tr>
<td></td>
<td>cParentJoinString (Character)</td>
</tr>
<tr>
<td></td>
<td>cChildJoinString (Character)</td>
</tr>
<tr>
<td></td>
<td>cParentJoinFields (Character)</td>
</tr>
<tr>
<td></td>
<td>cChildJoinFields (Character)</td>
</tr>
<tr>
<td></td>
<td>cParentReplace (Character)</td>
</tr>
<tr>
<td></td>
<td>cChildReplace (Character)</td>
</tr>
</tbody>
</table>

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.
Referential Integrity Manager API

The following API is the only public entry point for the Referential Integrity Manager.

versionData

This procedure coordinates versioning the data to be deployed.

Location:  ry\app\ryrisvrp.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, as outlined in the following sections:

- Overview
- Temp-tables used by Repository Managers
- Repository Manager APIs
- Repository Design Manager APIs
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 run-time tasks. The Repository Manager handles data transport for running applications. It is designed to efficiently return object definitions to applications at run time. 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 sets the global variable, server-side, to either YES for the server-side version or 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:

```plaintext
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>ry\app\ryrepclntp</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>ry\app\ryrepsrvrp.p</td>
</tr>
<tr>
<td>Main code</td>
<td>ry\app\ryrepmngrp.i</td>
</tr>
<tr>
<td>Included files</td>
<td>adm2\calltables.i</td>
</tr>
<tr>
<td></td>
<td>adm2\ttaction.i</td>
</tr>
<tr>
<td></td>
<td>adm2\tttoolbar.i</td>
</tr>
<tr>
<td></td>
<td>af\app\afdatatypi.i</td>
</tr>
<tr>
<td></td>
<td>af\app\afttsecurityctrl.i</td>
</tr>
<tr>
<td></td>
<td>af\app\aftttranslate.i</td>
</tr>
<tr>
<td></td>
<td>af\sup2\aferrortxt.i</td>
</tr>
<tr>
<td></td>
<td>af\sup2\afglobals.i</td>
</tr>
<tr>
<td></td>
<td>ry\app\rydefrescd.i</td>
</tr>
<tr>
<td></td>
<td>ry\app\rygetitemp.p</td>
</tr>
<tr>
<td></td>
<td>ry\app\rygetmensp.p</td>
</tr>
<tr>
<td></td>
<td>ry\app\rymenumfunc.i</td>
</tr>
<tr>
<td></td>
<td>ry\inc\ryattstori.i</td>
</tr>
<tr>
<td></td>
<td>ry\inc\getobjecti.i</td>
</tr>
<tr>
<td></td>
<td>ry\inc\ryrepset.i</td>
</tr>
<tr>
<td></td>
<td>ry\inc\ryrepgid1.i</td>
</tr>
</tbody>
</table>

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

Table 8–2:  Repository Design Manager files

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>ry\app\rydesclntp.p</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>ry\app\rydessrvrp.p</td>
</tr>
<tr>
<td>Main code</td>
<td>ry\app\rydesmngrp.i</td>
</tr>
<tr>
<td>Included files</td>
<td>adm2\calltables.i</td>
</tr>
<tr>
<td></td>
<td>adm2\globals.i</td>
</tr>
<tr>
<td></td>
<td>af\app\afdatatypi.i</td>
</tr>
<tr>
<td></td>
<td>af\sup2\afcheckerr.i</td>
</tr>
<tr>
<td></td>
<td>af\sup2\aferrortxt.i</td>
</tr>
<tr>
<td></td>
<td>ry\app\rydefrescd.i</td>
</tr>
<tr>
<td></td>
<td>ry\inc\rydestdefi.i</td>
</tr>
<tr>
<td></td>
<td>ry\inc\ryrepset.i</td>
</tr>
<tr>
<td></td>
<td>ry\inc\ryrepgid1.i</td>
</tr>
<tr>
<td>Other important files</td>
<td>ry\app\rygenomngp.p</td>
</tr>
</tbody>
</table>

For more information on these Managers, see the chapter on using the Progress Dynamics Managers in OpenEdge Development: Progress Dynamics Advanced Development.
Temp-tables used by Repository Managers

Because the Repository Managers coordinate data for the entire framework, they make extensive use of temp-tables. They reference several temp-tables defined elsewhere in the OpenEdge application development environment, including those in the following files:

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

Temp-tables used by Repository Manager

The Repository Manager uses the temp-tables described in Table 8–3 for object retrieval in the run-time environment. The transportClass temp-table is defined like the ttClass temp-table.

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

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttClass</td>
<td>ClassName (Character)</td>
</tr>
<tr>
<td>AND</td>
<td>ClassObj (Decimal)</td>
</tr>
<tr>
<td>transportClass</td>
<td>ClassTableName (Character)</td>
</tr>
<tr>
<td></td>
<td>ClassBufferHandle (Handle)</td>
</tr>
<tr>
<td></td>
<td>InheritsFromClass (Character)</td>
</tr>
<tr>
<td></td>
<td>SuperProcedures (Character)</td>
</tr>
<tr>
<td></td>
<td>SuperProcedureModes (Character)</td>
</tr>
<tr>
<td></td>
<td>SuperHandles (Character)</td>
</tr>
<tr>
<td></td>
<td>InstanceBufferHandle (Handle)</td>
</tr>
<tr>
<td></td>
<td>EventTableName (Character)</td>
</tr>
<tr>
<td></td>
<td>EventBufferHandle (Handle)</td>
</tr>
<tr>
<td></td>
<td>SetList (Character)</td>
</tr>
<tr>
<td></td>
<td>GetList (Character)</td>
</tr>
<tr>
<td></td>
<td>RuntimeList (Character)</td>
</tr>
<tr>
<td>ttEntity</td>
<td>EntityName (Character)</td>
</tr>
<tr>
<td></td>
<td>EntityTableName (Character)</td>
</tr>
<tr>
<td></td>
<td>EntityBufferHandle (Handle)</td>
</tr>
<tr>
<td></td>
<td>LanguageCode (Character)</td>
</tr>
<tr>
<td>Temp-table</td>
<td>Fields (data types)</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cacheObject</td>
<td>ObjectName (Character)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>cachePage</td>
<td>InstanceId (Decimal)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>cacheLink</td>
<td>InstanceId (Decimal)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ttEntityDump</td>
<td>tEntityName (Character)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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\ryrepset.i

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttStoreAttribute</td>
<td>tAttributeParent (Character)</td>
</tr>
<tr>
<td></td>
<td>tAttributeParentObj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>tAttributeLabel (Character)</td>
</tr>
<tr>
<td></td>
<td>tConstantValue (Logical)</td>
</tr>
<tr>
<td></td>
<td>tCharacterValue (Character)</td>
</tr>
<tr>
<td></td>
<td>tDecimalValue (Decimal)</td>
</tr>
<tr>
<td></td>
<td>tIntegerValue (Integer)</td>
</tr>
<tr>
<td></td>
<td>tDateValue (Date)</td>
</tr>
<tr>
<td></td>
<td>tRawValue (Raw)</td>
</tr>
<tr>
<td></td>
<td>tLogicalValue (Logical)</td>
</tr>
</tbody>
</table>

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\app\rymenufunc.i

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttBandToExtract</td>
<td>parent_menu_structure_code (Character)</td>
</tr>
<tr>
<td></td>
<td>menu_structure_type (Character)</td>
</tr>
<tr>
<td></td>
<td>menu_structure_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>menu_structure_code (Character)</td>
</tr>
<tr>
<td></td>
<td>extract_sequence (Integer)</td>
</tr>
<tr>
<td></td>
<td>disabled (Logical)</td>
</tr>
<tr>
<td></td>
<td>under_development (Logical)</td>
</tr>
<tr>
<td></td>
<td>menu_item_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>control_spacing (Integer)</td>
</tr>
<tr>
<td></td>
<td>control_padding (Integer)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Described in</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttTranslate</td>
<td>The “Temp-tables used by Localization Manager” section on page 5–3.</td>
</tr>
<tr>
<td>ttUser</td>
<td>The “Temp-tables used by General Manager” section on page 4–3.</td>
</tr>
</tbody>
</table>

### 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 on page 4–3.

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

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttSmartLink</td>
<td>tContainerObj (Decimal) tLinkName (Character) tUserLinkName (Character) tSourceObj (Decimal) tTargetObj (Decimal)</td>
</tr>
</tbody>
</table>
The Repository Design Manager APIs use the temp-tables described in Table 8–8 for generating objects.

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

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttDeleteAttribute1</td>
<td>tAttributeParent (Character) tAttributeParentObj (Decimal) tAttributeLabel (Character) tConstantValue (Logical) tCharacterValue (Character) tDecimalValue (Decimal) tIntegerValue (Integer) tDateValue (Date) tRawValue (Raw) tLogicalValue (Logical)</td>
</tr>
<tr>
<td>ttDataObjectField</td>
<td>tDatabaseName (Character) tDatabaseName (Character) tTableName (Character) tTableName (Character) tDumpName (Character) tDumpName (Character) tFieldName (Character) tFieldName (Character) tFieldOrder (Character) tFieldOrder (Character) tKeepField (Logical) tKeepField (Logical) tIsTableObjField (Logical) tIsTableObjField (Logical) tFieldUpdatable (Logical) tFieldUpdatable (Logical)</td>
</tr>
<tr>
<td>ttTableColumn</td>
<td>tDatabaseName (Character) tDatabaseName (Character) tTableName (Character) tTableName (Character) tDataColumns (Character) tDataColumns (Character) tUpdatableColumns (Character) tUpdatableColumns (Character) tWhereClause (Character) tWhereClause (Character) tOrder (Integer) tOrder (Integer)</td>
</tr>
<tr>
<td>ttRelate</td>
<td>cOwnerTable (Character) cOwnerTable (Character) cDataBaseName (Character) cDataBaseName (Character) cRelatedTable (Character) cRelatedTable (Character) cLinkFieldName (Character) cLinkFieldName (Character) cIndexName (Character) cIndexName (Character)</td>
</tr>
<tr>
<td>ttTableField</td>
<td>LIKE _field2 tDatabaseName (Character) tDatabaseName (Character) tTableName (Character) tTableName (Character) tTableDumpName (Character) tTableDumpName (Character) tKeyField (Logical) tKeyField (Logical) tEntityObjectField (Logical) tEntityObjectField (Logical)</td>
</tr>
</tbody>
</table>
Table 8–8: Temp-tables defined in ry\app\rygenomngp.p

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
</table>
| 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)  
|             | TableName (Character)  
|             | TableDumpName (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\app\rydesmngp.i**

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
</table>
| ttDeleteAttribute\textsuperscript{1} | tAttributeParent (Character)  
tAttributeParentObj (Decimal)  
tAttributeLabel (Character)  
tConstantValue (Logical)  
tCharacterValue (Character)  
tDecimalValue (Decimal)  
tIntegerValue (Integer)  
tDateValue (Date)  
tRawValue (Raw)  
tLogicalValue (Logical) |
| ttTableField | LIKE _field\textsuperscript{2}  
tDataBaseName (Character)  
tTableName (Character)  
tTableDumpName (Character)  
tKeyField (Logical)  
tEntityObjectField (Logical) |
| ttXref | tElementType (Character)  
tSourceData (Character)  
tTargetData (Character) |

\textsuperscript{1} This temp-table is defined like the ttStoreAttribute temp-table.

\textsuperscript{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.
Repository Manager APIs

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

**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\ryrepmngrp.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 filename for an include file associated with the object being parsed. If set to "clientProxy", the procedure calculates the `_cl client proxy` filename 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 an 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 filename based on the used input parameters. This can be used to create or access the physical filenames of the object being parsed or the associated file for the object.

OUTPUT pcError AS CHARACTER

Any errors encountered during processing that must 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\ryrepmngrp.i.

classHasAttribute

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

Location: ry\app\ryrepmngrp.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.
**classIsA**

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

**Location:** ry\app\ryngrp.

**Parameters:**
- INPUT pcClassName AS CHARACTER
- INPUT pcInheritsFromClass AS CHARACTER

**Returns:** LOGICAL

**Notes:** None

**Examples:** See the `generateCalculatedField` procedure in ry\app\rygenmgp.

**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\ryngrp.

**Parameters:** None

**Notes:** None

**Examples:** See the `cacheObjectOnClient` function in ry\app\ryngrp.

**createClassCache**

This procedure caches class attributes and UI events.

**Location:** ry\app\ryngrp.

**Parameters:**
- INPUT pcClassName AS CHARACTER

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

**Notes:** If this procedure uses the `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\ryngrp.
**destroyClassCache**

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

**Location:** `ry\app\ryrepmngrp.i`

**Parameters:** None

**Notes:** This is a separate API because in a run-time environment the class attributes are unlikely to change much.

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

---

**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\ryrepmngrp.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`.

---

**getCacheClassBuffer**

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

**Location:** `ry\app\ryrepmngrp.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 nonblank 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 wild card is passed in, the record might not be correctly repositioned.

**Examples:** See the `classHasAttribute` function in `ry\app\ryrepmngrp.i`.

---
**getCacheLinkBuffer**

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

**Location:** `ry\app\ryrepmngrp.i`

**Parameters:** None

**Returns:** HANDLE

**Notes:** None

**Examples:** See the `fetchUI` procedure in `ry\app\ruimsrvrp.p`.

**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\ryrepmngrp.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\ryrepmngrp.i`.

**getCachePageBuffer**

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

**Location:** `ry\app\ryrepmngrp.i`

**Parameters:** None

**Returns:** HANDLE

**Notes:** None

**Examples:** See the `fetchUI` procedure in `ry\app\ruimsrvrp.p`. 
**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\ryrepmngrp.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\aftermwizow.w`.

**getClassFromInstance**

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

**Location:** `ry\app\ryrepmngrp.i`

**Parameters:**

| INPUT | pdInstance AS DECIMAL |

The object’s Instance ID in the cache buffer.

**Returns:** CHARACTER

**Notes:** None

**getClientCacheDir**

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

**Location:** `ry\app\ryrepmngrp.i`

**Parameters:**

| INPUT | pcProductModule AS CHARACTER |

The product module.

| OUTPUT | pcFullPathName AS CHARACTER |

The full pathname to the directory.

**Notes:** None

**Examples:** See the `createClassCache` procedure in `ry\app\ryrepmngrp.i`. 
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 that is being launched.

Location:   ry\app\ryrepmngrp.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\ryrepmngrp.i.

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\ryrepmngrp.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\ryrepmngrp.i.
**getObjectNames**

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

**Location:** ry\app\ryrepmngrp.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\afsesmngrp.i.

**getObjectSuperProcedure**

This procedure returns an object’s custom super procedure to a caller.

**Location:** ry\app\ryrepmngrp.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\afsesmngrp.i.
**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`.

---

**IsA**

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

**Location:**  
`ry\app\ryrepmngrp.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`. 
plipShutdown

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

**Location:**  
ry\app\ryrepmngrp.i

**Parameters:**  
None

**Notes:**  
This procedure currently contains no active code.

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\ryrepmngrp.i

**Parameters:**

INPUT plDesignMode AS LOGICAL

INPUT-OUTPUT pcResultCodes AS CHARACTER

**Notes:**  
None

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

saveEntitiesToClientCache

This procedure saves cached entities to the disk. This procedure is called from the cache client generation tool as well as the session shutdown event.

**Location:**  
ry\app\ryrepmngrp.i

**Parameters:**

INPUT pcEntities AS CHARACTER

A comma-delimited list of entities to save to the Repository. If the value is blank, then all the currently cached entities are saved.

OUTPUT pcStatus AS CHARACTER

Status of the entity dump process. This is mainly required for the cache client tool.

**Notes:**  
None

**Examples:**  
See the ICFCFM_StartSessionShutdown procedure in ry\app\ryrepmngrp.i.
**startDataObject**

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

**Location:** `ry\app\ryrepmngrp.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\rydesmngrp.i`.

**storeAttributeValue**

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

**Location:** `ry\app\ryrepmngrp.i`

**Parameters:**

INPUT `phAttributeValueBuffer` AS HANDLE

INPUT TABLE-HANDLE `phAttributeValueTable`

**Notes:**

- This procedure does not allow the addition of attribute values to object instances that are contained by abstract classes.
- It is now possible to add and update attribute values for attributes marked as run time against a class.

**Examples:** See the `registerSdoFields` procedure in `ry\app\rydesmngrp.i`.
Repository Design Manager APIs

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

changeObjectInstance

This procedure changes the object instance.

**Location:**  ry\app\rydesmngp.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

Takes a valid result code, the name of a Master container, or ? (Unknown) for ALL.

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 plDeleteObjectSource AS LOGICAL

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

INPUT plRemoveDefaultAttributes AS LOGICAL

Removes default attributes set for the object instances.

INPUT plRemoveUnusedAttributes 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 trgInt\replace procedure in ry\prc\ryreplinstvp.p.

changeObjectType

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

Location: ry\app\rydesmngp.i

Parameters:

INPUT pcFileName AS CHARACTER

The object's filename.

INPUT pcObjectTypeCode AS CHARACTER

The new object type code.

INPUT plRemoveDefaultAttr 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 plRemoveNonOTAttr 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\ryrep\mngp.i.
classHasAttribute

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

**Location:**  ry\app\rydesmngrp.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 run-time attributes. Use this function in tools that must check design-time attributes.

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

clearDesignCache

This procedure empties the ttClassExtinformation temp-table.

**Location:**  ry\app\rydesmngrp.i

**Parameters:**  None

**Notes:**  None

**Examples:**  See the main block in ry\app\rydesmngrp.i.
copyObjectMaster

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

**Location:**  
`ry\app\rydesmngp.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.
**generateCalculatedField**

This procedure creates a calculated field.

**Location:**  
ry\app\rydesmngrp.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.

**generateClassCache**

This procedure outputs the class cache to the disk.

**Location:**  
ry\app\rydesmngrp.i

**Parameters:**

INPUT pcClassList AS CHARACTER  
A comma-delimited list of classes to output.

OUTPUT pcStatus AS CHARACTER  
The status of the class cache.

**Returns:**  
None

**Examples:**  
See the generateCache procedure in ry\obj\rycrunv.w.
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\rydesmngrp.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.
**generateDataLogicObject**

This procedure generates a DataLogic object procedure.

**Location:**  
ry\app\rydesmggrp.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 pcDataObjectName AS CHARACTER
- INPUT pcDataLogicRelativePath 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.
**generateDataObject**

This procedure generates SDOs and other data objects.

**Location:**  
ry\app\rydesmngrp.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.
**generateDynamicBrowse**

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

**Location:** ry\app\rydesmggp.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.
**generateDynamicSDF**

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

**Location:**  
`ry\app\rydesmngrp.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 attribute names from `pcAttributeLabels`.
- **INPUT pcAttributeDataType AS CHARACTER**  
  A CHR(1)-delimited list of attribute data type corresponding to attribute 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`.
**generateDynamicViewer**

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

**Location:**  
ry\app\rydesmngrp.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\rydesmgdv.i.

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

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

**Location:**  
`ry\app\rydesmngp.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`. 
generateEntityObject

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

Location: ry\app\rydesmngp.i

Parameters:

INPUT pcTableNames 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 pcEntityObjectType 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

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.

generateSBODataLogicObject

This procedure generates the Data Logic Procedure (DLP) for the SBO and its client proxy, compiles the files, and registers the DLP in the Repository.

Location: ry\app\rydesmngrp.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 the 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 the new logic procedure to be created.

INPUT pcLogicObjectType AS CHARACTER

The logic type of the 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 the 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 plCreateMissingFolder AS LOGICAL

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

Notes: None
Examples: See the main block in ry\app\rydesgsdlp.p.

**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 plDeleteExistingInstances 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.
**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\rydesmngrp.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.
**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`.

**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:

```plaintext
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:**

INPUT `pcValueField` AS CHARACTER

The field from the `product_module` table used as the value field in the LIST-ITEM-PAIRS.

INPUT `pcDescFields` AS CHARACTER

A comma-delimited list of product module fields to display in the label portion of the LIST-ITEM-PAIRS. This can be a maximum of three fields.

INPUT `pcDescFormat` AS CHARACTER

A base string containing substitution parameters of the form `&N` used to substitute the description fields.

INPUT `pcDelimiter` AS CHARACTER

A 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 determine whether or not to show Repository modules.
- This function replaces the deprecated productModuleList function.

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

getschemaqueryhandle

This function creates a query for the metaschema.

Location: ry\app\rydesmngrp.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\rydesmngrp.i.

getwidgetsizeofformat

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

Location: ry\app\rydesmngrp.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\rydesmngrp.i.
insertClass

This procedure adds a class to the Repository or modifies an existing class.

**Location:**  ry\app\rydesmngrp.p

**Parameters:**

INPUT pcClassName AS CHARACTER

The unique class name. This is a mandatory parameter. Blanks are not allowed. If the class passed in is not found, it is created in the Repository.

INPUT pcClassDescription AS CHARACTER

A description of the class. Blanks are not allowed.

INPUT pcExtendsClassName AS CHARACTER

The name of the class extended by the class given in pcClassName. Blank clears the value. The class specified must already exist in the Repository.

INPUT pcCustomClassName AS CHARACTER

The name of the class that customizes the class given in pcClassName. The custom class cannot inherit from the same class that pcClassName does. Blank clears the value. The class specified must already exist in the Repository.

INPUT plClassDisabled AS LOGICAL

Whether or not the class is disabled.

INPUT plLayoutSupported AS LOGICAL

Whether or not layouts are supported for this class.

INPUT pcDeploymentType AS CHARACTER

The default deployment types for objects of this class. Can be one or more of the following: “Cli” for client, “Srv” for server, “Web,” or blank.

INPUT plStaticObject AS LOGICAL

Whether or not the class contains static objects.

INPUT plCacheOnClient AS LOGICAL

Whether or not procedure code can be generated and deployed onto a client.

INPUT pcClassObjectName AS CHARACTER

This parameter is not used.

OUTPUT pdClassObj AS DECIMAL

The obj field of the class being maintained.
Notes:

- Unless otherwise specified in the parameter description, all parameters are nonmandatory, blank values are allowed, and an Unknown value (?) keeps the existing value in the Repository.

- This procedure does not update attributes, UI events, and supported links for a class. They are updated through other APIs.

**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 p1ForceCreateNew 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\rydesmgp.d

**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\rydesmgp.d.
**insertObjectLinks**

This procedure adds or updates an object’s links.

**Location:**  
`ry\app\rydesmngrp.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`.

**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\rydesmngrp.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.

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.

Examples: See the generateDataFields procedure in ry\app\rydesmngp.i.
**insertObjectPage**

This procedure creates and updates an object page.

**Location:**  ry\app\rydesmgp\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 must 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.
**insertSupportedLink**

This procedure adds or updates a supported link record for a class.

**Location:**  
ry\app\rydesmngrp.p

**Parameters:**

INPUT pcClassName AS CHARACTER  
The name of the class for which the link is supported. This is a mandatory parameter. Blanks and Unknown value (?) are not allowed.

INPUT pcLinkName AS CHARACTER  
The name of the link being added. Only valid links from the smartlink_type table are allowed. This is a mandatory parameter. Blanks and Unknown value (?) are not allowed.

INPUT plDeactivateOnHide AS LOGICAL  
Whether or not this type of link is automatically deactivated when the object is hidden. If YES, the link is deactivated when the object is hidden, and activated when the object is viewed. The Unknown value (?) keeps the existing value.

INPUT plIsLinkSource AS LOGICAL  
Whether or not this type of object can be a source for the specified link. If YES, the object can be the source for the link. The Unknown value (?) keeps the existing value.

INPUT plIsLinkTarget AS LOGICAL  
Whether or not this type of object can be a target for the specified link. If YES, the object can be the target for the link. The Unknown value (?) keeps the existing value.

OUTPUT pdSupportedLinkObj AS DECIMAL  
The object id of the updated record.

**Notes:**  
None
**insertUiEvents**

This procedure stores UI events.

**Location:** ry\app\rydesmngrp.i

**Parameters:**

- INPUT phUiEventBuffer AS HANDLE
- INPUT TABLE-HANDLE phUiEventTable

**Notes:**

- The buffer used in this procedure is based on the ttStoreUiEvent temp-table defined in ry\inc\ryrepset.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.

**ObjectExists**

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

**Location:** ry\app\rydesmngrp.i

**Parameters:**

- INPUT pcObjectName AS CHARACTER

**Returns:** LOGICAL

**Notes:** None

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

**plipShutdown**

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

**Location:** ry\app\rydesmngrp.i

**Parameters:** None

**Notes:** This procedure currently contains no active code.
**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\rydesmngrp.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, nonblank, 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 box calls this function using pcAction = 'Default'.
Examples: See the insertObjectMaster procedure in ry\app\rydesmngrp.i.

removeAttributeValues

This procedure removes attribute values.

Location: ry\app\rydesmngrp.i

Parameters:

INPUT phAttributeValueBuffer AS HANDLE
INPUT TABLE-HANDLE phAttributeValueTable

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

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

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\rydesmngrp.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.

**removeObjectPage**

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

**Location:** \ry\app\rydesmngrp.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 plRemoveObjectInstances 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
**removePageInstance**

This procedure removes a page object record.

**Location:**  
ry\app\rydesmngp.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\rydesmngp.i.

**removeSupportedLink**

This procedure removes a supported link record for a class.

**Location:**  
ry\app\rydesmngp.p

**Parameters:**

- **INPUT pcClassName AS CHARACTER**  
The name of the class for which the link is supported. This is a mandatory parameter. Blanks and Unknown value (?) are not allowed.
- **INPUT pcLinkName AS CHARACTER**  
The name of the link being removed. Only valid links from the smartlink_type table are allowed. This is a mandatory parameter. Blanks and Unknown value (?) are not allowed.

**Notes:**  
None
**removeUIEvents**

This procedure removes UI Events.

**Location:**  
`ry\app\rydesmngrp.i`

**Parameters:**

- INPUT `phUiEventBuffer` AS HANDLE
- INPUT TABLE-HANDLE `phUiEventTable`

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

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

**setQualifiedTableName**

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

**Location:**  
`ry\app\rydesmngrp.i`

**Parameters:**

- INPUT `plSuppressDbName` AS LOGICAL

**Returns:**  
LOGICAL

**Notes:**  
None
The Security Manager handles retrieving and applying security information from the Repository, as outlined in the following sections:

- Overview
- Temp-Tables used by Security Manager
- Security Manager APIs
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 entities 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 sets the global variable, server-side, to either YES for the server-side version or NO for the client-side version. Both programs then include the main code of the manager from the af\app\afsecmngrp.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>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>af\sup2\afsecc1ntp.p</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>af\app\afsecsrvrp.p</td>
</tr>
<tr>
<td>Main code</td>
<td>af\app\afsecmngrp.i</td>
</tr>
<tr>
<td>Included files</td>
<td>af\app\afsegtsecctr1p.i</td>
</tr>
<tr>
<td></td>
<td>af\app\afsecttddef.i</td>
</tr>
<tr>
<td></td>
<td>af\app\afttsecurityctr1.i</td>
</tr>
<tr>
<td></td>
<td>af\app\afsecupusrallocp.i</td>
</tr>
<tr>
<td></td>
<td>af\app\usrSecChck.i</td>
</tr>
<tr>
<td></td>
<td>af\sup2\aferrortxt.i</td>
</tr>
<tr>
<td></td>
<td>af\sup2\afglobals.i</td>
</tr>
</tbody>
</table>

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in *OpenEdge Development: Progress Dynamics Advanced Development*.

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 af\app\afsecmngrp.i

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttUserSecurityCheck</td>
<td>dUserObj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>dOrganisationObj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>cOwningEntityMnemonic (Character)</td>
</tr>
<tr>
<td></td>
<td>dOwningObj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>lSecurityCleared (Logical)</td>
</tr>
<tr>
<td></td>
<td>CSecurityValue1 (Character)</td>
</tr>
<tr>
<td></td>
<td>CSecurityValue2 (Character)</td>
</tr>
<tr>
<td>ttFieldSecurityCheck</td>
<td>cObjectName (Character)</td>
</tr>
<tr>
<td></td>
<td>cAttributeNameCode (Character)</td>
</tr>
<tr>
<td></td>
<td>cSecurityOptions (Character)</td>
</tr>
<tr>
<td>ttTokenSecurityCheck</td>
<td>cObjectName (Character)</td>
</tr>
<tr>
<td></td>
<td>cAttributeNameCode (Character)</td>
</tr>
<tr>
<td></td>
<td>cSecurityOptions (Character)</td>
</tr>
<tr>
<td>ttTableSecurityCheck</td>
<td>cOwningEntityMnemonic (Character)</td>
</tr>
<tr>
<td></td>
<td>cEntityFieldname (Character)</td>
</tr>
<tr>
<td></td>
<td>cValidValues (Character)</td>
</tr>
<tr>
<td>ttRangeSecurityCheck</td>
<td>cRangeCode (Character)</td>
</tr>
<tr>
<td></td>
<td>cObjectName (Character)</td>
</tr>
<tr>
<td></td>
<td>cAttributeNameCode (Character)</td>
</tr>
<tr>
<td></td>
<td>cRangeFrom (Character)</td>
</tr>
<tr>
<td></td>
<td>cRangeTo (Character)</td>
</tr>
<tr>
<td>ttObjectSecurityCheck</td>
<td>cObjectName (Character)</td>
</tr>
<tr>
<td></td>
<td>dObjectObj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>lRestricted (Logical)</td>
</tr>
<tr>
<td>ttMenuSecurity</td>
<td>cName (Character)</td>
</tr>
<tr>
<td></td>
<td>cMenuType (Character)</td>
</tr>
<tr>
<td></td>
<td>cSecurityOptions (Character)</td>
</tr>
<tr>
<td>ttUser</td>
<td>userObj (Decimal)</td>
</tr>
<tr>
<td>ttGlobalSecurityStructure</td>
<td>product_module_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>owning_entity_mnemonic (Character)</td>
</tr>
<tr>
<td></td>
<td>owning_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>security_structure_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>security_object_name (Character)</td>
</tr>
<tr>
<td></td>
<td>restricted (Logical)</td>
</tr>
<tr>
<td></td>
<td>user_allocation_value1 (Character)</td>
</tr>
<tr>
<td></td>
<td>user_allocation_value2 (Character)</td>
</tr>
<tr>
<td>ttGlobalSecurityAllocation</td>
<td>loginorganisation_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>owning_entity_mnemonic (Character)</td>
</tr>
<tr>
<td></td>
<td>owning_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>user_allocation_value1 (Character)</td>
</tr>
<tr>
<td></td>
<td>user_allocation_value2 (Character)</td>
</tr>
</tbody>
</table>
The Security Manager uses the temp-table described in Table 9–3 to cache security setting information.

**Table 9–3: Temp-table defined in af\app\afsecttdef.i**

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttUpdatedAllocations</td>
<td>lDeleteAll (Logical)</td>
</tr>
<tr>
<td></td>
<td>lDelete (Logical)</td>
</tr>
<tr>
<td></td>
<td>lUpdateValue1AndValue2 (Logical)</td>
</tr>
<tr>
<td></td>
<td>owning_entity_mnemonic (Character)</td>
</tr>
<tr>
<td></td>
<td>owning_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>user_allocation_value1 (Character)</td>
</tr>
<tr>
<td></td>
<td>user_allocation_value2 (Character)</td>
</tr>
</tbody>
</table>

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-table defined in af\app\afttsecurityctrl.i**

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttSecurityControl</td>
<td>security_control_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>password_max_retries (Integer)</td>
</tr>
<tr>
<td></td>
<td>password_history_life_time (Integer)</td>
</tr>
<tr>
<td></td>
<td>full_access_by_default (Logical)</td>
</tr>
<tr>
<td></td>
<td>security_enabled (Logical)</td>
</tr>
<tr>
<td></td>
<td>help_writer_enabled (Logical)</td>
</tr>
<tr>
<td></td>
<td>build_top_menus_only (Logical)</td>
</tr>
<tr>
<td></td>
<td>default_help_filename (Character)</td>
</tr>
<tr>
<td></td>
<td>error_log_filename (Character)</td>
</tr>
<tr>
<td></td>
<td>translation_enabled (Logical)</td>
</tr>
<tr>
<td></td>
<td>login_filename (Character)</td>
</tr>
<tr>
<td></td>
<td>multi_user_check (Logical)</td>
</tr>
<tr>
<td></td>
<td>program_access_check (Logical)</td>
</tr>
<tr>
<td></td>
<td>minimise_siblings (Logical)</td>
</tr>
<tr>
<td></td>
<td>enable_window_positioning (Logical)</td>
</tr>
<tr>
<td></td>
<td>force_unique_password (Logical)</td>
</tr>
<tr>
<td></td>
<td>company_logo_filename (Character)</td>
</tr>
<tr>
<td></td>
<td>system_icon_filename (Character)</td>
</tr>
<tr>
<td></td>
<td>small_icon_filename (Character)</td>
</tr>
<tr>
<td></td>
<td>product_logo_filename (Character)</td>
</tr>
<tr>
<td></td>
<td>scm_logo_filename (Character)</td>
</tr>
<tr>
<td></td>
<td>scm_tool_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>user_context_expiry_period (Decimal)</td>
</tr>
</tbody>
</table>
Security Manager APIs

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

areFieldsCached

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

Location: af\app\afsecmngrp.i
Parameters:
  INPUT pcObjectName AS CHARACTER
  INPUT pcRunAttribute AS CHARACTER
Returns: LOGICAL
Notes: None
Examples: See the containerCacheUpfront procedure in af\app\afsesmngrp.i.

areTokensCached

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

Location: af\app\afsecmngrp.i
Parameters:
  INPUT pcObjectName AS CHARACTER
  INPUT pcRunAttribute AS CHARACTER
Returns: LOGICAL
Notes: None
Examples: See the containerCacheUpfront procedure in af\app\afsesmngrp.i.
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\afsecmngrp.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 dynamics\as_connect.p.

cacheGlobalSecurityAllocations

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

**Location:** af\app\afsecmngrp.i

**Parameters:** None

**Notes:** None

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

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\afsecmngrp.i

**Parameters:** None

**Notes:** None

**Examples:** See the clearClientCache procedure in af\app\afsecmngrp.i.
**changePassword**

This procedure changes a user’s password, making all the relevant checks to the password. 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\afsecmngrp.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 piLength AS INTEGER**
  
  The number of password characters entered in the new password.

- **INPUT plExpired AS LOGICAL**
  
  The password expired flag.

- **OUTPUT pcError AS CHARACTER**
  
  Any failure reason (standard Dynamics-formatted error).

**Notes:** None

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

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\afsecmngrp.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.

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\afsecmngrp.i

Parameters: None

Notes: None

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

createGroupAllocation

This procedure adds a group allocation for a user.

Location: af\app\afsecmngrp.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\afsecmngrp.i.
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\afsecmngrp.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\secusrgrpp.p`.

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\afsecmngrp.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 `pFieldSecurity` 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\afsecmngrp.i.

fieldSecurityCheck

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

Location: af\app\afsecmngrp.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.

Notes: None
Examples: See the widgetWalk procedure in af\app\afsecmngrp.i.

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\afsecmngrp.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\afsesmngrp.i.

**getContainerIcons**

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

Location: af\app\afsecmngrp.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.

**getFieldSecurity**

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

Location: af\app\afsecmngrp.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 that 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\gstadvview.w.
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\afsecmmgrp.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.

getSecurityControl

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

Location:       af\app\afsecmmgrp.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\afsesmmgrp.i.
menuItemSecurityCheck

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

**Location:** af\app\afsecmngrp.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.

menuStructureSecurityCheck

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

**Location:** af\app\afsecmngrp.i

**Parameters:**

INPUT pcStructure AS CHARACTER

The menu structures (bands) to check.

OUTPUT plStructureHidden 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.
objectSecurityCheck

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

Location: af\app\afsecmngrp.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 plSecurityRestricted 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.

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\afsecmngrp.i

Parameters: None

Notes: This procedure currently contains no active code.
rangeSecurityCheck

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

**Location:** af\app\afsecmngp.i

**Parameters:**

INPUT pcRangeCode AS CHARACTER

The range code to check for a user security clearance.

INPUT pcObjectName AS CHARACTER

The current program object for a security check.

INPUT pcAttributeCode AS CHARACTER

The instance attribute posted to the program.

OUTPUT pcRangeFrom AS CHARACTER

The FROM value permitted for a user, blank for all.

OUTPUT pcRangeTo AS CHARACTER

The TO value permitted for a user, blank for all.

**Notes:** None

receiveCacheSessionSecurity

This procedure receives the initial session security cache.

**Location:** af\app\afsecmngp.i

**Parameters:**

INPUT TABLE FOR ttSecurityControl

**Notes:** None

**Examples:** See the loginCacheAfter procedure in af\app\afsesmngp.i.
tableSecurityCheck

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

Location: af\app\afsecmngp.i

Parameters:

INPUT pcOwningEntityMnemonic AS CHARACTER

The table FLA for which to check a 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

tokenSecurityCheck

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

Location: af\app\afsecmngp.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\afsecmngp.i.
**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\afsecmngrp.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 settings.

**Notes:**  
None

**Examples:**  
See the `widgetWalk` procedure in `af\app\afsesmngrp.i`. 
updateUserAllocations

This procedure creates user security allocations in the Repository.

**Location:** af\app\afsecmngrp.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.

userLoginOrganisations

This procedure checks which organizations a user can access.

**Location:** af\app\afsecmngrp.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.
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\afsecmngrp.i

**Parameters:**

- INPUT pdUserObj AS DECIMAL
  - The user being checked.
- INPUT pdOrganisationObj AS DECIMAL
  - The company the user is logged in to.
- INPUT pcOwningEntityMnemonic AS CHARACTER
  - The security table being checked.
- INPUT pdOwningObj AS DECIMAL
  - The security table object being checked.
- INPUT plReturnValues 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, as outlined in the following sections:

- Overview
- Temp-tables used by Session Manager
- Session Manager APIs
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 sets the global variable, server-side, to either YES for the server-side version or NO for the client-side version. Both programs then include the main code of the manager from the af\app\afsesmngrp.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**

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>af\sup2\afsesclntp.p</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>af\app\afsessrvrp.p</td>
</tr>
<tr>
<td>Main code</td>
<td>af\app\afsesmngrp.i</td>
</tr>
</tbody>
</table>

**Included files**

- adec\comm\appsrvtt.i
- adec\comm\appserv.i
- adm2\callttables.i
- adm2\callttparam.i
- adm2\globals.i
- adm2\ttaction.i
- adm2\tttoolbar.i
- af\app\afsesgetglcp.i
- af\app\afttglobalctrl.i
- af\app\afttpersist.i
- af\app\afttprofiledata.i
- af\app\afttsecurityctrl.i
- af\app\afttttranslate.i
- af\app\aftttranslation.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\afttcombo.i
- ry\app\rydefrescd.i

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in *OpenEdge Development: Progress Dynamics Advanced Development.*
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 OpenEdge application development environment, including those in the following files:

- adecomm\appsrvtt.i
- adm2\calltables.i
- adm2\callttparam.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 af\applafsesmngrp.i**

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttProperty</td>
<td>propertyName (Character)</td>
</tr>
<tr>
<td></td>
<td>propertyValue (Character)</td>
</tr>
<tr>
<td>ttPersistProc</td>
<td>hProc (Handle)</td>
</tr>
<tr>
<td>ttActionUnderway</td>
<td>action_underway_origin (Character)</td>
</tr>
<tr>
<td></td>
<td>action_table_fla (Character)</td>
</tr>
<tr>
<td></td>
<td>action_type (Character)</td>
</tr>
<tr>
<td></td>
<td>action_primary_key (Character)</td>
</tr>
<tr>
<td></td>
<td>action_scn_object_name (Character)</td>
</tr>
<tr>
<td>ttViewerCol</td>
<td>dColumn (Decimal)</td>
</tr>
<tr>
<td></td>
<td>dColWidth (Decimal)</td>
</tr>
<tr>
<td></td>
<td>dMaxLabel (Decimal)</td>
</tr>
<tr>
<td></td>
<td>dNewCol (Decimal)</td>
</tr>
</tbody>
</table>
The Session Manager uses the temp-table described in Table 10–3 to cache login information.

**Table 10–3: Temp-table defined in afapp\logintt.i**

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttLoginUser</td>
<td>encoded_user_name (Character)</td>
</tr>
<tr>
<td></td>
<td>default_organisation_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>language_obj (Decimal)</td>
</tr>
</tbody>
</table>

The Session Manager uses the temp-table described in Table 10–4, a copy of the Repository gsc_global_control table, to cache information about system-wide defaults.

**Table 10–4: Temp-table defined in afapp\aftttglobalctrl.i**

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttGlobalControl</td>
<td>global_control_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>default_country_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>default_nationality_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>default_language_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>default_currency_obj (Decimal)</td>
</tr>
<tr>
<td></td>
<td>date_format (Character)</td>
</tr>
<tr>
<td></td>
<td>date_format_mask (Character)</td>
</tr>
<tr>
<td></td>
<td>repository_site_number (Integer)</td>
</tr>
</tbody>
</table>

The Session Manager uses the temp-tables described in Table 10–5 to cache information about persistently running procedures in the current session.

**Table 10–5: Temp-tables defined in afapp\afttpersist.i**

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Fields (data types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttPersistentProc</td>
<td>physicalName (Character)</td>
</tr>
<tr>
<td></td>
<td>logicalName (Character)</td>
</tr>
<tr>
<td></td>
<td>runAttribute (Character)</td>
</tr>
<tr>
<td></td>
<td>childDataKey (Character)</td>
</tr>
<tr>
<td></td>
<td>onAppserver (Logical)</td>
</tr>
<tr>
<td></td>
<td>procedureType (Character)</td>
</tr>
<tr>
<td></td>
<td>procedureHandle (Handle)</td>
</tr>
<tr>
<td></td>
<td>runPermanent (Logical)</td>
</tr>
<tr>
<td></td>
<td>multiInstanceSupported (Logical)</td>
</tr>
<tr>
<td></td>
<td>currentOperation (Character)</td>
</tr>
<tr>
<td></td>
<td>uniqueId (Integer)</td>
</tr>
<tr>
<td></td>
<td>startDate (Date)</td>
</tr>
<tr>
<td></td>
<td>startTime (Integer)</td>
</tr>
<tr>
<td></td>
<td>procedureVersion (Character)</td>
</tr>
<tr>
<td></td>
<td>procedureNarration (Character)</td>
</tr>
<tr>
<td></td>
<td>internalEntries (Character)</td>
</tr>
</tbody>
</table>

| ttProcDependancy| parentProcedureHandle (Handle)                          |
|                | childProcedureHandle (Handle)                           |
|                | parentType (Character)                                  |
|                | childType (Character)                                   |
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**

<table>
<thead>
<tr>
<th>Temp-table</th>
<th>Described in</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttEntityMnemonic</td>
<td>The “Temp-tables used by General Manager” section on page 4–3.</td>
</tr>
<tr>
<td>ttProfileData</td>
<td>The “Temp-tables used by Profile Manager” section on page 6–3.</td>
</tr>
<tr>
<td>ttTranslation</td>
<td>The “Temp-tables used by Localization Manager” section on page 5–3.</td>
</tr>
<tr>
<td>ttUser</td>
<td>The “Temp-tables used by General Manager” section on page 4–3.</td>
</tr>
</tbody>
</table>

**Session Manager APIs**

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

**activateSession**

This procedure sets up the record for a remote session in the Repository’s `gst_session` table.

**Location:**  af\app\afsesmngp.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:** This procedure’s calls set the appropriate session parameters. It then calls the `establishSession` procedure, which actually reactivates the session.

**Examples:** See the main block in dynamics\as Activate.p.
addAsSuperProcedure

This function adds a procedure and all of its super procedures to a specified target procedure.

Location: af\app\afsesmngrp.i

Parameters:

INPUT phSuperProcedure AS HANDLE

The procedure to be added as a super procedure.

INPUT phProcedure AS HANDLE

The target procedure.

Returns: LOGICAL

Notes: None

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

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 suppressDisplay property 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 box. 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\afsesmngrp.i

Parameters:

INPUT pcMessageList AS CHARACTER
INPUT pcButtonList AS CHARACTER
INPUT pcDefaultButton AS CHARACTER
INPUT pcCancelButton AS CHARACTER
INPUT pcDialogTitle 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\fullobjcw.w.

contextHelp

This procedure serves as the context help launcher to aid in the Progress Dynamics context help integration.

Location: af\app\afsesmngrp.i

Parameters:

INPUT phObject AS HANDLE
    The handle of the object containing the 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.
createLinks

This procedure aids in creating links between objects.

**Location:** af\app\afsesmngrp.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\afsesmngrp.i.

deletePersistentProc

This procedure deletes persistent procedures started since the Session Manager started.

**Location:** af\app\afsesmngrp.i

**Parameters:** None

**Notes:** None

**Examples:** See the `plipShutdown` procedure in af\app\afsesmngrp.i.

deleteSession

This procedure deletes a session and all its context from the Repository.

**Location:** af\app\afsesmngrp.i

**Parameters:**

- INPUT `pcSessionID` AS CHARACTER
  
  The session ID to delete.

**Notes:** None

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

This procedure is an update to the activateSession procedure. Before this procedure is called, the following session properties should have been set, either by a call to the storeAppServerInfo procedure or manually:

- client_DynamicsVersion
- client_PhysicalSessType
- client_NumericSeparator
- client_DecimalPoint
- client_DateFormat
- client_YearOffset
- client_SessionType
- client_OldSessionID
- client_NewSessionID

**Location:** af\app\afsesmngrp.i

**Parameters:**

INPUT plReactivate AS LOGICAL

INPUT plConfirmExpired S LOGICAL

**Notes:** None

**Examples:** See the activateSession procedure in af\app\afsesmngrp.i.
**getActionUnderway**

This procedure gets the values for fields in the `ttActionUnderway` temp-table for the passed-in record.

**Location:** `af\app\afsesmngp.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\fullobjcw.w`.

**getCurrentLogicalName**

This function returns the value of the `gcLogicalContainerName` variable set in the Session Manager's `launchContainer` procedure.

**Location:** `af\app\afsesmngp.i`

**Parameters:** None

**Returns:** CHARACTER

**Notes:** None
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\afsesmngrp.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.

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\afsesmngrp.i

Parameters:

INPUT-OUTPUT TABLE-HANDLE hHelpTable

Notes: None

Examples: See the buildTempTable procedure in ry\obj\rydynhelpw.v.
getInternalEntryExists

This function checks whether a procedure or function exists for a given handle.

Location: af\app\afsesmngp.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 an OpenEdge 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\afsesmngp.i.

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\afsesmngp.i
Parameters:
- OUTPUT TABLE FOR ttLoginUser
Notes: None
Examples: See the main block in af\app\cachelogin.p.

getPersistentProcs

This procedure retrieves a temp-table, ttPersistentProc, of running persistent procedures. The temp-table is used to make the information available for display in a browser outside the Session Manager.

Location: af\app\afsesmngp.i
Parameters:
- OUTPUT TABLE FOR ttPersistentProc
  - The temp-table of persistent procedures.
Notes: None
getPropertyList

This function retrieves properties from either the local temp-table or the Repository through the server-side Session Manager.

The function returns a CHR(3)-delimited list of corresponding property values.

Location: af\app\afsesmngp.i

Parameters:

INPUT pcPropertyList AS CHARACTER

A comma-delimited list of properties for which to retrieve values.

INPUT plSessionOnly AS LOGICAL

Indicates how to retrieve the properties. If YES, retrieve the value from the client session. If NO, retrieve the value from the client session if available, otherwise retrieve from the server. If unknown (?), retrieve from the server only.

Returns: CHARACTER

Notes:

• The returned list of properties always has as many entries as pcPropertyList. If a property in pcPropertyList does not have a value, the procedure returns a blank value for that property.

• If plSessionOnly is set to YES, the Repository is never checked. Unless the property value is set in the cache, a blank is returned even if a server value exists for the property.

• If plSessionOnly is set to NO, the cache value is returned, if possible. If no cache value exists, the server value is retrieved and returned. If no value exists in the server, a blank is returned as the value of the property.

• If plSessionOnly is set to Unknown value (?), the cache is never checked and the value is read from the server. If no server value has been set, a blank is returned as the value of the property, even if there is a client cache value.

Examples: See the askQuestion procedure in af\app\afsesmngp.i.
getSessionList

This procedure retrieves a comma-delimited list of all the session types in the Repository.

Location: af\app\afsesmngrp.i

Parameters:

INPUT pcTypes AS CHARACTER

This parameter is reserved for future use.

OUTPUT pcSessionList AS CHARACTER

A comma-delimited list of session types.

Notes: None

Examples: See the populateSelectionList procedure in af\obj2\gsmsebconv.w.

getSessTypeOverrideInfo

This procedure gets the session type override information from the Repository.

Location: af\app\afsesmngrp.i

Parameters:

OUTPUT plConfigDefault AS LOGICAL

Can the session types in the Repository be overridden?

OUTPUT plAllowUnregistered AS LOGICAL

Can users start sessions with unregistered session types?

OUTPUT pcPermittedUnregistered AS CHARACTER

A comma-delimited list of session types that are not defined in the Repository that are allowed to access the Repository.

Notes: This procedure wrappers the getSessTypeOverrideInfo procedure in af/app/afsesstypecachep.p.

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

This procedure displays the “About” help message. The procedure uses an ADM showMessage call using the passed-in container so that all the object names and versions are shown in the system information.

**Location:** af\app\afsesmngrp.i

**Parameters:**

INPUT phContainer AS HANDLE

The container procedure handle.

**Notes:** None

**Examples:** See the buildMenus procedure in af\cod2\afmenumaintw.w.

helpContents

This procedure displays the help contents from the help file.

**Location:** af\app\afsesmngrp.i

**Parameters:**

INPUT phContainer AS HANDLE

The container procedure handle.

**Notes:** None

**Examples:** See the buildMenus procedure in af\cod2\afmenumaintw.w.

helpHelp

This procedure displays the help contents from the help file.

**Location:** af\app\afsesmngrp.i

**Parameters:**

INPUT phContainer AS HANDLE

The container procedure handle.

**Notes:** None

**Examples:** See the buildMenus procedure in af\cod2\afmenumaintw.w.
helpTopics

This procedure displays the help contents from the help file.

**Location:** af\app\afsesmngp.i

**Parameters:**

INPUT phContainer AS HANDLE

The container procedure handle.

**Notes:** None

**Examples:** See the buildMenus procedure in af\cod2\afmenumaintw.w.

htmlHelpKeywords

This procedure displays the help topics using keyword lookup from an HTML help file.

**Location:** af\app\afsesmngp.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
**htmlHelpTopic**

This procedure displays a specific help topic in an HTML help file.

**Location:** af\app\afsesmngrp.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\afsesmngrp.i.

**increaseFrameforPopup**

This procedure increases the width of a frame for a pop-up window. If necessary, it also increases the window width.

**Location:** af\app\afsesmngrp.i

**Parameters:**

- **INPUT phObject AS HANDLE**

- **INPUT phFrame AS HANDLE**

- **INPUT phLookup AS HANDLE**

- **INPUT phWidget AS HANDLE**

**Notes:** Replaces increaseFrameWidth procedure.
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 using 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 must be replaced with a decimal point to ensure that the query resolves properly.

Location: af\app\afsesmngrp.i
Parameters:

INPUT pcQueryString AS CHARACTER

The query being examined.

INPUT piPosition AS INTEGER

Returns: LOGICAL
Notes: None

killPlips

This procedure shuts down PLIPs cleanly, forcing the correct update of the running procedure’s temp-table.

Location: af\app\afsesmngrp.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 must be specified, 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\rycusmngrp.i.
**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\afsesmngrp.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 `plOnAppserver` AS LOGICAL

If running on the AppServer, set to YES.

**Notes:** None

**Examples:** See the main block in ry\app\ryplipshut.i.
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\afsesmngrp.i

Parameters:

INPUT pcObjectFileName AS CHARACTER

The object filename. If it is not known, the physical name or the physical and logical names together can be used.

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 plOnceOnly 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 to run or the running object.

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.

LaunchExternalProcess

This procedure launches an external process.

Location: af\app\afsesmngrp.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.
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\afsesmngrp.i

Parameters:

INPUT pcPhysicalName AS CHARACTER
    The physical object filename, with path and extension.

INPUT plOnceOnly 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 plRunPermanent AS LOGICAL
    The default is NO.

OUTPUT phProcedureHandle AS HANDLE
    The procedure handle of object run.

Notes:

- If the plRunPermanent flag is set to YES, then this procedure is not automatically killed when an AppServer agent is deactivated. Ordinarily this flag should be NO, and all procedures left running should be deleted at the end of an AppServer request by the deactivation routine. 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 shut down incorrectly for some reason.
• If the pcOnAppserver flag is set to APPSERVER, then this procedure can only be run on AppServer. If the flag is YES and no AppServer partition is passed in, then it defaults to the AstraAppServer and the session handle, gshAstraAppserver handle, is used for the AppServer. If any other partition is passed in and the flag is not NO, then the partition is connected if required. Any partitions connected in this manner are disconnected by the shutdown procedure, af\sup2\afshutdwnp.p.

• Generally, you do not need to run this procedure for most of the Progress Dynamics Managers because their handles are available through the systemwide global shared variables. However, in order to add the Managers to the temp-table of running persistent procedures, this procedure is used for the Managers when they are first run.

Examples: See the launchClassObject function in ry\app\ryrepmngrp.i.

notifyUser

This procedure notifies the user of some message by some means, for example, e-mail.

Location: af\app\afsesmngrp.i

Parameters:

INPUT pdUserObj AS DECIMAL

The object number of the user record to notify.

INPUT pcUserName AS CHARACTER

The user name of the 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, “e-mail.”

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\afsesmngrp.i.
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 shut down 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\afsesmngrp.i
**Parameters:** None
**Notes:** None

resizeLookupFrame

This procedure resizes a lookup SDF frame to fit new labels.

**Location:** af\app\afsesmngrp.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\afsesmngrp.i.
**resizeNormalFrame**

This procedure resizes a standard frame to fit new labels. SDF frames do not use this procedure.

**Location:** af\app\afsesmngrp.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\afsesmngrp.i.

**resizeSDFFrame**

This procedure resizes an SDF frame to fit new labels.

**Location:** af\app\afsesmngrp.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\afsesmngrp.i.
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\afsesmngp.i

Parameters:

INPUT phFocus AS HANDLE

The handle of the focused widget.

Notes: None

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

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\afsesmngp.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\afsesmngp.i.

setActionUnderway

This procedure sets the values in the ttActionUnderway temp-table for the passed-in record.

Location: af\app\afsesmngp.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\fulloobjcw.w.
**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\afsesmngrp.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\afsesmngrp.i`.

---

**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\afsesmngrp.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 `relogon` procedure in `af\app\afsesmngrp.i`. 
**setReturnValue**

This procedure returns whatever was sent in to set the required RETURN-VALUE.

**Location:** af\app\afsesmngrp.i

**Parameters:**

- **INPUT pcReturnValue AS CHARACTER**
  
  The required return value.

**Notes:** None

**setSecurityForDynObjects**

This function sets security properties for Dynamic Lookups and Dynamic Combos.

**Location:** af\app\afsesmngrp.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\afsesmngrp.i.

**setSessTypeOverrideInfo**

This procedure sets the session type override information from the Repository.

**Location:** af\app\afsesmngrp.i

**Parameters:**

- **OUTPUT plConfigDefault AS LOGICAL**
  
  Can the session types in the Repository be overridden?

- **OUTPUT p1AllowUnregistered AS LOGICAL**
  
  Can users start sessions with unregistered session types?

- **OUTPUT pcPermittedUnregistered AS CHARACTER**
  
  A comma-delimited list of session types that are not defined in the Repository that are allowed to access the Repository.
Notes: This procedure verifies whether a user is connected with a valid session type and sets these values only if the type is valid.

Examples: See the applyChanges procedure in af\obj2\gsmsctrlv.w.

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 suppressDisplay property 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 ADM 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 box, af\cod2\afmsgaged.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\afsesmngrp.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 p1DisplayEmpty 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\aftermlognw.w.

**showWarningMessages**

This procedure issues a warning to a user without generating an input-blocking statement in the process.

Location:  af\app\afsesmngrp.i  

Parameters:  

INPUT pcMessageList AS CHARACTER  

One or more messages using the standard formatting from aferroptxt.i.

INPUT pcMessageType AS CHARACTER  

For example, ERR (Error) or INF (Information).

INPUT pcMessageTitle AS CHARACTER  

The title of the message dialog box.

Notes:  None
translateWidgets

This procedure translates widget labels. It is called from the widgetWalk procedure.

**Location:** af\app\afsesmngrp.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\afsesmngrp.i.

updateErrorLog

This procedure updates the messages into the error log database table.

**Location:** af\app\afsesmngrp.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\afsesmngrp.i.

updateHelp

This procedure updates the help table with the supplied temp-table.

**Location:** af\app\afsesmngrp.i

**Parameters:**

- INPUT TABLE-HANDLE hHelpTable

**Notes:** None

**Examples:** See the updateRecord procedure in ry\obj\rydynhelpv.w.
widgetWalk

This procedure walks the widget tree for the passed-in frame.

**Location:** af\app\afsesmngp.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 plPopupsInFields 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 needed to create and return the DHTML to fulfill client requests received by the Web Request Manager, as outlined in the following sections:

- Overview
- User Interface Manager APIs
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 first retrieves session data from the Session Manager and layout data from the Repository Manager. It then generates and sends the DHTML to the client. The UI Manager then fetches the necessary application data and sends it to the client. That is, the UI Manager sends HTML script and data separately.

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>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Main code</td>
<td><code>ry\app\ryuimsrvrp.p</code></td>
</tr>
<tr>
<td>Included files</td>
<td><code>adm2\ttaction.i</code></td>
</tr>
<tr>
<td></td>
<td><code>adm2\tttoolbar.i</code></td>
</tr>
<tr>
<td></td>
<td><code>adm2\treettdef.i</code></td>
</tr>
<tr>
<td></td>
<td><code>af\sup\afghplipdf.i</code></td>
</tr>
<tr>
<td></td>
<td><code>af\sup2\afglobals.i</code></td>
</tr>
<tr>
<td></td>
<td><code>ry\app\rydefrescd.i</code></td>
</tr>
<tr>
<td></td>
<td><code>ry\app\ryobjretri.i</code></td>
</tr>
<tr>
<td></td>
<td><code>ry\app\ryuimbl.i</code></td>
</tr>
<tr>
<td></td>
<td><code>ry\incl\lval.i</code></td>
</tr>
<tr>
<td></td>
<td><code>web2\wrap-cgi.i</code></td>
</tr>
<tr>
<td></td>
<td><code>wrappers\lognote.i</code></td>
</tr>
</tbody>
</table>

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in *OpenEdge Development: Progress Dynamics Advanced Development*. 
User Interface Manager APIs

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

### escapeData

This function will escape data line single quote, new line, and carriage return characters.

**Location:** ry\app\ryuimbl.i

**Parameters:**

pcData AS CHARACTER

**Returns:** CHARACTER

**Notes:** None

**Examples:** See the ouputConflicndata function in ry\app\ryuimbl.i.

### 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\ryuimbl.i

**Parameters:**

INPUT cAction AS CHARACTER

The action to apply.

**Notes:** None

**Examples:** See the setMessage procedure in ry\app\ryuimbl.i.
Web Request Manager

The Web Request Manager is the single point of entry for all Web requests, as outlined in the following sections:

- Overview
- Web Request Manager API
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>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Server-side wrapper</td>
<td>NA</td>
</tr>
<tr>
<td>Main code</td>
<td>ry\app\ryreqsrvrp.p</td>
</tr>
<tr>
<td>Included files</td>
<td>af\app\afttsecurityctrl.i</td>
</tr>
<tr>
<td></td>
<td>af\sup\afghplipdf.i</td>
</tr>
<tr>
<td></td>
<td>af\sup2\afglobals.i</td>
</tr>
<tr>
<td></td>
<td>web2\wrap-cgi.i</td>
</tr>
<tr>
<td></td>
<td>wrappers\lognote.i</td>
</tr>
</tbody>
</table>

For more information on this Manager, see the chapter on using the Progress Dynamics Managers in *OpenEdge Development: Progress Dynamics Advanced Development*.

Web Request Manager API

The following API is the only public entry point for the Web Request Manager.

**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

**Examples:** See the run-web-object procedure in ry\app\ryreqsrvrp.p.
Index

A
activateSession procedure
   Session Manager 10–6
addAsSuperProcedure function
   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
   getASConnectionString function 2–13
   getConnectionParams function 2–13
   getConnectionString function 2–14
   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 2–16
   loadPartitionInfo procedure 2–17
   parseConnectionParams function 2–17
   plipShutdown procedure 2–17
   reconnectService procedure 2–17
   registerService procedure 2–18
   savePartitionInfo procedure 2–18
   security_prompt procedure 2–18
   setServiceHandle function 2–18
appServerConnect procedure
   AppServer Connection Manager 2–11
appServerDisconnect procedure
   AppServer Connection Manager 2–11
areFieldsCached function
   Security Manager 9–5
areTokensCached function
   Security Manager 9–5
askQuestion procedure
   Session Manager 10–7
authenticateUser procedure
   Security Manager 9–6
B
buildClientCache procedure
   Localization Manager 5–4
   Profile Manager 6–3
buildWidgetTable procedure
   Localization Manager 5–4
C
calculateObjectPaths procedure
   Repository Manager 8–11
changeObjectInstance procedure
   Repository Design Manager 8–22
changeObjectType procedure
   Repository Design Manager 8–23
changePassword procedure
   Security Manager 9–7
checkIfOverlaps procedure
  General Manager 4–5
checkProfileDataExists procedure
  Profile Manager 6–4
checkUser procedure
  Security Manager 9–8
classHasAttribute function
  Repository Design Manager 8–24
  Repository Manager 8–12
classIsA function
  Repository Manager 8–13
clearClientCache procedure
  Localization Manager 5–5
  Profile Manager 6–4
  Repository Manager 8–13
  Security Manager 9–9
clearDesignCache procedure
  Repository Design Manager 8–24
Configuration File Manager
detectFileType function 1–4
expandTokens function 1–5
findFile function 1–5
getCodePath function 1–5
getExpandablePropertyValue function 1–6
getManagerHandle function 1–6
getPhysicalSessionType function 1–6
getProcedureHandle function 1–7
initializeSession procedure 1–7
isConfigManRunning function 1–8
isICFRunning function 1–8
obtainCFMTables procedure 1–8
obtainRegistryKeys procedure 1–9
parseConfig procedure 1–9
setPropertyManager procedure 1–9
sessionShutdown procedure 1–10
setICFIsRunning function 1–10
setSessionParam function 1–10
subscribeAll procedure 1–11
Connection Manager
connectService procedure 2–5
connectServiceWithParams procedure 2–5, 2–12
disconnectService procedure 2–6
findServiceRecord function 2–13
getConnectionParams function 2–6
getConnectionString function 2–6
getPhysicalService function 2–6
getServiceHandle function 2–7
getServiceList function 2–7
getServiceSTManager function 2–7
getServiceType function 2–7
getServiceTypeManager function 2–8
iisServiceRegistered function 2–9
initializeServices procedure 2–8
isConnected function 2–8
obtainConnectionTables procedure 2–9
plipShutdown procedure 2–9
reconnectService procedure 2–10
registerService procedure 2–10
setServiceTypeManager function 2–10
contextHelp procedure
  Session Manager 10–8
convertTimeToInteger function
  General Manager 4–6
copyObjectMaster procedure
  Repository Design Manager 8–25
createClassCache procedure
  Repository Manager 8–13
createFolder function
  General Manager 4–7
createGroupAllocation procedure
  Security Manager 9–9
createGroupFromUser procedure
  Security Manager 9–10
createLinks procedure
  Session Manager 10–9
Custom mangers
  creating Service Type Managers 2–2
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
getClientResultCodes function 3–5
InitializeObject procedure 3–5
PlipShutdown procedure 3–6
Index
Index

setClientResultCodes procedure 3–6
setSessionResultCodes procedure 3–6

D
Database Connection Manager
connectService procedure 2–19
connectServiceWithParams procedure 2–19
disconnectService procedure 2–19
findServiceRecord function 2–20
getConnectionRecord function 2–20
getConnectionString function 2–21
getServiceField function 2–21
getServiceHandle function 2–21
getServiceList function 2–22
isConnected function 2–22
isDefaultService function 2–22
parseConnectionParams function 2–23
plipShutdown procedure 2–23
registerService procedure 2–23
setServiceHandle function 2–24

definedPartitions function
AppServer Connection Manager 2–12
deletePersistentProc procedure
Session Manager 10–9
deleteSession procedure
Session Manager 10–9

defleteSessionProfile procedure
Profile Manager 6–5
destroyClassCache procedure
Repository Manager 8–14
detectFileType function
Configuration File Manager 1–4
disconnectService procedure
AppServer Connection Manager 2–12
Connection Manager 2–6
Database Connection Manager 2–19
WebService Service Type Manager 2–25

E
escapeData function
UI Manager 11–3
establishSession procedure
Session Manager 10–10
expandTokens function
Configuration File Manager 1–5
extractRootFile procedure
Repository Manager 8–14

F
fieldandtokenSecurityCheck procedure
Security Manager 9–10
fieldSecurityCheck procedure
Security Manager 9–11
fieldSecurityGet procedure
Security Manager 9–11
findFile function
Configuration File Manager 1–5
findServiceRecord function
Connection Manager 2–13
Database Connection Manager 2–20
WebService Service Type Manager 2–26
formatPersonDetails function
General Manager 4–7

g
General Manager
checkIfOverlaps procedure 4–5
convertTimeToInteger function 4–6
createFolder function 4–7
formatPersonDetails function 4–7
getDBsForImportedEntities procedure 4–7
getDBVersion procedure 4–8
getDumpName procedure 4–8
gGetEntityCacheBuffer function 4–9
gGetEntityDescription procedure 4–9
gGetEntityDetail procedure 4–10
gGetEntityDisplayField procedure 4–10
gGetEntityExists procedure 4–11
gGetEntityFieldCacheBuffer function 4–11
gGetEntityTableName procedure 4–11
gGetHighKey function 4–12
gGetInternalEntries function 4–12
gGetKeyField function 4–12
gGetLanguageText procedure 4–13
gGetNextSequenceValue function 4–14
gGetObjField function 4–14
gGetOEMCode function 4–15
gGetOEMDescription function 4–15
gGetPropertyFromList function 4–15
gGetRecordCheckAudit procedure 4–16
gGetRecordCheckComment procedure 4–16
gGetRecordCheckDetail procedure 4–17
gGetRecordUserProp procedure 4–17
gGetSequenceConfirmation function 4–17
gGetSequenceExist procedure 4–18
gGetSequenceMask procedure 4–18
gGetSiteNumber procedure 4–18
gGetStatusObj function 4–19
gGetStatusShortDesc function 4–19

Index–3
Index

getStatusTLA function 4–19
ggetTableDumpName function 4–20
ggetTableInfo procedure 4–20
ggetTableInfoObj procedure 4–21
ggetUpdatableTableInfo function 4–21
ggetUpdatableTableInfoObj function 4–22
getUserSourceLanguage procedure 4–22
haveOutstandingUpdates function 4–22
listLookup function 4–23
plipShutdown procedure 4–23
refreshMnemonicsCache procedure 4–23
setPropertyValueInList function 4–24
setWidgetAttribute function 4–24
updateTableViaSDO procedure 4–25
validateEntityMnemonic procedure 4–26
generateCalculatedField procedure
Repository Design Manager 8–26
generateClassCache procedure
Repository Design Manager 8–26
generateDataFields procedure
Repository Design Manager 8–27
generateDataLogicObject procedure
Repository Design Manager 8–28
generateDataObject procedure
Repository Design Manager 8–29
generateDynamicBrowse procedure
Repository Design Manager 8–30
generateDynamicSDF procedure
Repository Design Manager 8–31
generateDynamicViewer procedure
Repository Design Manager 8–32
generateEntityInstances procedure
Repository Design Manager 8–33
generateEntityObject procedure
Repository Design Manager 8–34
generateSBODataLogicObject procedure
Repository Design Manager 8–35
generateSDOInstances procedure
Repository Design Manager 8–36
generateVisualObject procedure
Repository Design Manager 8–37
getActionUnderway procedure
Session Manager 10–11
getASCConnectionString function
AppServer Connection Manager 2–13
getBufferDbName function
Repository Design Manager 8–38
getCacheClassBuffer function
Repository Manager 8–14
getCacheLinkBuffer function
Repository Manager 8–15
getCacheObjectBuffer function
Repository Manager 8–15
getCachePageBuffer function
Repository Manager 8–15
getEntityCacheBuffer function
General Manager 4–9
getEntityDescription procedure
General Manager 4–9
getEntityDetail procedure
    General Manager 4–10
getEntityDisplayField procedure
    General Manager 4–10
getEntityExists procedure
    General Manager 4–11
getEntityFieldCacheBuffer function
    General Manager 4–11
getEntityTableName procedure
    General Manager 4–11
getExpandablePropertyValue function
    Configuration File Manager 1–6
getAddressSecurity procedure
    Security Manager 9–12
getAddressSecurity function
    General Manager 4–12
getAddressEntries function
    General Manager 4–12
getAddressEntryExists function
    Session Manager 10–13
getAddressPartitions function
    AppServer Connection Manager 2–14
getAddressPtnInfo function
    AppServer Connection Manager 2–14
getAddressField function
    General Manager 4–12
getAddressLanguageText procedure
    General Manager 4–13
getAddressLoginUserInfo procedure
    Session Manager 10–13
getAddressManagerHandle function
    Configuration File Manager 1–6
getAddressMandatoryTables procedure
    Security Manager 9–13
getAddressMappedFilename function
    Repository Manager 8–17
getAddressNextSequenceValue function
    General Manager 4–14
getAddressObjectNames procedure
    Repository Manager 8–18
getObjectSuperProcedure procedure
    Repository Manager 8–18
getObjectField function
    General Manager 4–14
getOEMCode function
    General Manager 4–15
getOEMDescription function
    General Manager 4–15
getPartitionsByType function
    AppServer Connection Manager 2–14
getAddressPersistentProcs procedure
    Session Manager 10–13
getAddressPhysicalService function
    AppServer Connection Manager 2–15
    Connection Manager 2–6
    Database Connection Manager 2–21
    WebService Service Type Manager 2–27
getAddressPhysicalSessionType function
    Configuration File Manager 1–6
getAddressProcedureHandle function
    Configuration File Manager 1–7
getAddressProductModuleList function
    Repository Design Manager 8–38
getAddressProfileData procedure
    Profile Manager 6–5
getAddressProfileTTHandle function
    Profile Manager 6–6
getAddressPropertyFromList function
    General Manager 4–15
getAddressPropertyList function
    Session Manager 10–14
getAddressRecordCheckAudit procedure
    General Manager 4–16
getAddressRecordCheckComment procedure
    General Manager 4–16
getAddressRecordDetail procedure
    General Manager 4–17
getAddressRecordUserProp procedure
    General Manager 4–17
getAddressReferenceLanguage function
    Customization Manager 3–3
getAddressReferenceLoginCompany function
    Customization Manager 3–4
getAddressReferenceSystem function
    Customization Manager 3–4
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>getReferenceUIType function</td>
<td>Customization Manager 3–4</td>
</tr>
<tr>
<td>getReferenceUser function</td>
<td>Customization Manager 3–4</td>
</tr>
<tr>
<td>getReferenceUserCategory function</td>
<td>Customization Manager 3–5</td>
</tr>
<tr>
<td>getSchemaQueryHandle function</td>
<td>Repository Design Manager 8–39</td>
</tr>
<tr>
<td>getSecurityControl procedure</td>
<td>Security Manager 9–13</td>
</tr>
<tr>
<td>getSequenceConfirmation function</td>
<td>General Manager 4–17</td>
</tr>
<tr>
<td>getSequenceExist procedure</td>
<td>General Manager 4–18</td>
</tr>
<tr>
<td>getSequenceMask procedure</td>
<td>General Manager 4–18</td>
</tr>
<tr>
<td>getServiceField function</td>
<td>AppServer Connection Manager 2–15</td>
</tr>
<tr>
<td>getServiceField function</td>
<td>Database Connection Manager 2–21</td>
</tr>
<tr>
<td>getServiceField function</td>
<td>WebService Service Type Manager 2–27</td>
</tr>
<tr>
<td>getServiceHandle function</td>
<td>AppServer Connection Manager 2–15</td>
</tr>
<tr>
<td>getServiceHandle function</td>
<td>Connection Manager 2–7</td>
</tr>
<tr>
<td>getServiceHandle function</td>
<td>Database Connection Manager 2–21</td>
</tr>
<tr>
<td>getServiceHandle function</td>
<td>WebService Service Type Manager 2–27</td>
</tr>
<tr>
<td>getServiceList function</td>
<td>AppServer Connection Manager 2–16</td>
</tr>
<tr>
<td>getServiceList function</td>
<td>Connection Manager 2–7</td>
</tr>
<tr>
<td>getServiceList function</td>
<td>Database Connection Manager 2–22</td>
</tr>
<tr>
<td>getServiceList function</td>
<td>WebService Service Type Manager 2–28</td>
</tr>
<tr>
<td>getServiceSTManager function</td>
<td>Connection Manager 2–7</td>
</tr>
<tr>
<td>getServiceType function</td>
<td>Connection Manager 2–7</td>
</tr>
<tr>
<td>getServiceTypeManager function</td>
<td>Connection Manager 2–8</td>
</tr>
<tr>
<td>getSessionCustomisationReferences function</td>
<td>Customization Manager 3–5</td>
</tr>
<tr>
<td>getSessionList procedure</td>
<td>Session Manager 10–15</td>
</tr>
<tr>
<td>getSessionParam function</td>
<td>Configuration File Manager 1–7</td>
</tr>
<tr>
<td>getSessionResultCodes function</td>
<td>Customization Manager 3–5</td>
</tr>
<tr>
<td>getSessTypeOverrideInfo procedure</td>
<td>Session Manager 10–15</td>
</tr>
<tr>
<td>getSiteNumber procedure</td>
<td>General Manager 4–18</td>
</tr>
<tr>
<td>getStatusObj function</td>
<td>General Manager 4–19</td>
</tr>
<tr>
<td>getStatusShortDesc function</td>
<td>General Manager 4–19</td>
</tr>
<tr>
<td>getStatusTLA function</td>
<td>General Manager 4–19</td>
</tr>
<tr>
<td>getTableDumpName function</td>
<td>General Manager 4–20</td>
</tr>
<tr>
<td>getTableInfo procedure</td>
<td>General Manager 4–20</td>
</tr>
<tr>
<td>getTableInfoObj procedure</td>
<td>General Manager 4–21</td>
</tr>
<tr>
<td>getToolbarBandActions procedure</td>
<td>Repository Manager 8–19</td>
</tr>
<tr>
<td>getTranslation procedure</td>
<td>Localization Manager 5–5</td>
</tr>
<tr>
<td>getUpdatableTableInfo function</td>
<td>General Manager 4–21</td>
</tr>
<tr>
<td>getUpdatableTableInfoObj function</td>
<td>General Manager 4–22</td>
</tr>
<tr>
<td>getUserSourceLanguage procedure</td>
<td>General Manager 4–22</td>
</tr>
<tr>
<td>getWidgetSizeFromFormat function</td>
<td>Repository Design Manager 8–39</td>
</tr>
</tbody>
</table>

**H**

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>haveOutstandingUpdates function</td>
<td>General Manager 4–22</td>
</tr>
<tr>
<td>helpAbout procedure</td>
<td>Session Manager 10–16</td>
</tr>
<tr>
<td>helpContents procedure</td>
<td>Session Manager 10–16</td>
</tr>
<tr>
<td>helpHelp procedure</td>
<td>Session Manager 10–16</td>
</tr>
<tr>
<td>helpTopics procedure</td>
<td>Session Manager 10–17</td>
</tr>
<tr>
<td>htmlHelpKeywords procedure</td>
<td>Session Manager 10–17</td>
</tr>
<tr>
<td>htmlHelpTopic procedure</td>
<td>Session Manager 10–18</td>
</tr>
</tbody>
</table>
increaseFrameForPopup procedure
Session Manager 10–18

InitializeObject procedure
Customization Manager 3–5

initializeSelf procedure
AppServer Connection Manager 2–16

initializeServices procedure
Connection Manager 2–8

initializeSession procedure
Configuration File Manager 1–7

insertClass procedure
Repository Design Manager 8–40

insertObjectInstance procedure
Repository Design Manager 8–41

insertObjectLinks procedure
Repository Design Manager 8–42

insertObjectMaster procedure
Repository Design Manager 8–42

insertObjectPage procedure
Repository Design Manager 8–44

insertSupportedLink procedure
Repository Design Manager 8–45

insertUiEvents procedure
Repository Design Manager 8–46

IsA function
Repository Manager 8–19

isConfigManRunning function
Configuration File Manager 1–8

isConnected function
AppServer Connection Manager 2–16
Connection Manager 2–8
Database Connection Manager 2–22
Web Service Service Type Manager 2–28

isLoggedInService function
AppServer Connection Manager 2–16
Database Connection Manager 2–22
Web Service Service Type Manager 2–28

isICFRunning function
Configuration File Manager 1–8

isObjQuoted function
Session Manager 10–19

isServiceRegistered function
Connection Manager 2–9

killPlips procedure
Session Manager 10–19

killProcedure procedure
Session Manager 10–20

launchContainer procedure
Session Manager 10–21

LaunchExternalProcess procedure
Session Manager 10–22

launchProcedure procedure
Session Manager 10–23

listLookup function
General Manager 4–23

loadPartitionInfo procedure
AppServer Connection Manager 2–17

Localization Manager
buildClientCache procedure 5–4
buildWidgetTable procedure 5–4
clearClientCache procedure 5–5
getTranslation procedure 5–5
multiTranslation procedure 5–6
plipShutdown procedure 5–7
receiveCacheClient procedure 5–7
translatePhrase function 5–8
translateSingleObject procedure 5–8
translateWidgetTable procedure 5–9
updateTranslations procedure 5–9

menuItemSecurityCheck procedure
Security Manager 9–14

menuStructureSecurityCheck procedure
Security Manager 9–14

multiTranslation procedure
Localization Manager 5–6

notifyUser procedure
Session Manager 10–24
**O**

ObjectExists function  
Repository Design Manager 8–46

objectSecurityCheck procedure  
Security Manager 9–15

obtainCFMTables procedure  
Configuration File Manager 1–8

obtainConnectionTables procedure  
Connection Manager 2–9

obtainRegistryKeys procedure  
Configuration File Manager 1–9

**P**

parseConfig procedure  
Configuration File Manager 1–9

parseConnectionParams function  
AppServer Connection Manager 2–17  
Database Connection Manager 2–23  
WebService Service Type Manager 2–28

PlipShutdown procedure  
Customization Manager 3–6

plipShutdown procedure  
AppServer Connection Manager 2–17  
Connection Manager 2–9  
Database Connection Manager 2–23  
General Manager 4–23  
Localization Manager 5–7  
Profile Manager 6–6  
Repository Design Manager 8–46  
Repository Manager 8–20  
Security Manager 9–15  
Session Manager 10–25  
WebService Service Type Manager 2–28

processRequest procedure  
Web Request Manager 12–2

Profile Manager  
buildClientCache procedure 6–3  
checkProfileDataExists procedure 6–4  
clearClientCache procedure 6–4  
deleteSessionProfile procedure 6–5  
getProfileData procedure 6–5  
getProfileTTHandle function 6–6  
plipShutdown procedure 6–6  
receiveProfileCache procedure 6–6  
setProfileData procedure 6–7  
updateCacheToDb procedure 6–8

propertyExpander procedure  
Configuration File Manager 1–9

**R**

rangeSecurityCheck procedure  
Security Manager 9–16

receiveCacheClient procedure  
Localization Manager 5–7

receiveCacheSessionSecurity procedure  
Security Manager 9–16

receiveProfileCache procedure  
Profile Manager 6–6

reconnectService procedure  
AppServer Connection Manager 2–17  
Connection Manager 2–10  
WebService Service Type Manager 2–29

Referential Integrity Manager  
versionData procedure 7–4

refreshMnemonicsCache procedure  
General Manager 4–23

registerService procedure  
AppServer Connection Manager 2–18  
Connection Manager 2–10  
Database Connection Manager 2–23

removeAttributeValues procedure  
Repository Design Manager 8–48

removeObject procedure  
Repository Design Manager 8–48

removeObjectPage procedure  
Repository Design Manager 8–49

removePageInstance procedure  
Repository Design Manager 8–50

removeSupportedLink procedure  
Repository Design Manager 8–50

removeUIEvents procedure  
Repository Design Manager 8–51

Repository Design Manager  
changeObjectInstance procedure 8–22  
changeObjectType procedure 8–23  
classHasAttribute function 8–24  
clearDesignCache procedure 8–24  
copyObjectMaster procedure 8–25  
generateCalculatedField procedure 8–26  
generateClassCache procedure 8–26  
generateDataFields procedure 8–27  
generateDataLogicObject procedure 8–28  
generateDataObject procedure 8–29  
generateDynamicBrowse procedure 8–30  
generateDynamicSDF procedure 8–31  
generateDynamicViewer procedure 8–32
Index

contextHelp procedure 10–8
createLinks procedure 10–9
deletePersistentProc procedure 10–9
deleteSession procedure 10–9
establishSession procedure 10–10
getActionUnderway procedure 10–11
getCurrentLogicalName procedure 10–11
getGlobalControl procedure 10–12
getHelp procedure 10–12
getInternalEntryExists function 10–13
getLoginUserInfo procedure 10–13
getPersistentProcs procedure 10–13
getPropertyList function 10–14
getSessionList procedure 10–15
getSessTypeOverrideInfo procedure 10–15
helpAbout procedure 10–16
helpContents procedure 10–16
helpHelp procedure 10–16
helpTopics procedure 10–17
htmlHelpKeywords procedure 10–17
htmlHelpTopic procedure 10–18
increaseFrameforPopup procedure 10–18
isObjQuoted function 10–19
killPlips procedure 10–19
killProcedure procedure 10–20
launchContainer procedure 10–21
LaunchExternalProcess procedure 10–22
launchProcedure procedure 10–23
notifyUser procedure 10–24
plipShutdown procedure 10–25
resizeLookupFrame procedure 10–25
resizeNormalFrame procedure 10–26
resizeSDFFrame procedure 10–26
runLookup procedure 10–27
sendEmail procedure 10–27
setActionUnderway procedure 10–28
setAttributesInObject procedure 10–29
setPropertyList function 10–29
setReturnValue procedure 10–30
setSecurityForDynObjects function 10–30
setSessTypeOverrideInfo procedure 10–30
setClientAction procedure
UI Manager 11–3
setClientResultCodes procedure
Customization Manager 3–6
setICFIsRunning function
Configuration File Manager 1–10
setProfileData procedure
Profile Manager 6–7
setPropertyList function
Session Manager 10–29
setPropertyValueInList function
General Manager 4–24
setQualifiedTableName function
Repository Design Manager 8–51
setReturnValue procedure
Session Manager 10–30
setServiceHandle function
AppServer Connection Manager 2–18
Database Connection Manager 2–24
WebService Service Type Manager 2–30
setServiceTypeManager function
Connection Manager 2–10
setSessionParam function
Configuration File Manager 1–10
setSessionResultCodes procedure
Customization Manager 3–6
setSessTypeOverrideInfo procedure
Session Manager 10–30
setWidgetAttribute function
General Manager 4–24
showMessages procedure
Session Manager 10–31
showWarningMessages procedure
Session Manager 10–32
startDataObject procedure
Repository Manager 8–21
storeAttributeValues procedure
Repository Manager 8–21
subscribeAll procedure
Configuration File Manager 1–11
T

tableSecurityCheck procedure  
Security Manager 9–17

tokenSecurityCheck procedure  
Security Manager 9–17

tokenSecurityGet procedure  
Security Manager 9–18

translatePhrase function  
Localization Manager 5–8

translateSingleObject procedure  
Localization Manager 5–8

translateWidgets procedure  
Session Manager 10–33

translateWidgetTable procedure  
Localization Manager 5–9

U

updateCacheToDb procedure  
Profile Manager 6–8

updateErrorLog procedure  
Session Manager 10–33

updateHelp procedure  
Session Manager 10–33

updateTableViaSDO procedure  
General Manager 4–25

updateTranslations procedure  
Localization Manager 5–9

updateUserAllocations procedure  
Security Manager 9–19

User Interface Manager

escapeData function 11–3

setClientAction procedure 11–3

userLoginOrganisations procedure  
Security Manager 9–19

userSecurityCheck procedure  
Security Manager 9–20

userLoginOrganisations procedure  
Security Manager 9–19

userLoginOrganisations procedure  
Security Manager 9–19

userSecurityCheck procedure  
Security Manager 9–20

V

validateEntityMnemonic procedure  
General Manager 4–26

versionData procedure  
Referential Integrity Manager 7–4

W

Web Request Manager

processRequest procedure 12–2

WebService Service Type Manager

connectService procedure 2–24

connectServiceWithParams procedure 2–25

disconnectService procedure 2–25

findServiceRecord function 2–26

getConnectionParams function 2–26

getConnectionString function 2–26

getServiceField function 2–27

getServiceHandle function 2–27

getServiceList function 2–28

isConnected function 2–28

isDefaultService function 2–28

parseConnectionParams function 2–28

plipShutdown procedure 2–29

reconnectService procedure 2–29

setServiceHandle function 2–30

widgetWalk procedure  
Session Manager 10–34